

Blind Signature

Musfirah Mohd Ali WEK020147

Supervisor : Mr. Yamani Idna Idris

Moderator: Mrs. Rafidah Md Noor

Abstract

In this thesis the author introduce a cryptographic construct called Blind Signature. This thesis is enhanced from digital signature system. The aim for this project is to develop reliable software which is capable to protect the data integrity and authenticity using blind signature method. Blind signature allow user to digitally sign the document without knowing the contents of the document. The signature has a blindness property, so if the signer later sees a document he has signed he will not be able to determine when or for whom he signed it.

The organization of this report is divided into five chapters. The first chapter is about the objectives, motivations, scope and limitation of this project. Then, in order to construct the blind signature system that used an appropriate kind of method, the author did a deep research and comparison on the existing cryptography, programming languages, algorithms and systems which are explained in chapter two of this project. Beside, the author also used the software methodology model as a guidance to produce a report smoothly and on time. The chosen methodology is explained in chapter 3. In chapter 4 of this project explained the system requirements which are functional and non-functional requirements. This chapter also enlightened what kind of scheme that the author chose for this system. Finally, the system design is developed. In this chapter 5 the overall architecture of the system, the data flow diagram and the system interface prototype is presented.

Acknowledgement

First and foremost, I would like to thank my advisor En Yamani Idna Idris for his invaluable source of help while conducting the research that eventually evolved into this thesis. He helped the author to generate the idea and spent a considerable amount of time making sure that the author had a deep understanding on the research.

Additionally, the author also would like to thank to anyone who had contributed directly or indirectly during the progress of this project.

University of Malaysia

Table of Contents

| | |
|---|----------|
| Abstract | i |
| Acknowledgements | ii |
| Table of Contents | iii |
| List of Illustrations | vi |
| Chapter 1: Introduction | 1 |
| 1.1 Aim | 1 |
| 1.2 Problems and Motivation | 1 |
| 1.3 Objectives | 2 |
| 1.4 Project scope | 3 |
| 1.5 Limitations | 3 |
| Chapter 2: Literature Review | 5 |
| 2.1 The Background of Cryptography | 7 |
| 2.2 Cryptographic Primitives | 7 |
| 2.2.1 Block Ciphers | 8 |
| 2.2.2 Additives Stream Ciphers | 10 |
| 2.2.3 Cryptographic Hash Functions | 10 |
| 2.3 Secret Key Cryptography | 11 |
| 2.3.1 Feistel Cipher Structure | 12 |
| 2.3.2 Symmetric Encryption Algorithm | 15 |
| 2.3.2.1 Data Encryption Standard | 15 |
| 2.3.2.2 International Data Encryption Algorithm | 17 |
| 2.3.2.3 Blowfish | 18 |
| 2.4 Public Key Cryptography | 18 |
| 2.4.1 Public Key Algorithm | 21 |
| 2.4.1.1 Rivest, Shamir, Adleman (RSA) | 21 |
| 2.4.1.2 Digital Signature Algorithm (DSA) | 23 |
| 2.5 Hash Functions | 24 |
| 2.5.1 One Way Hash Funtions | 26 |
| 2.5.1.1 Secure Hash Algorithm (SHA-1) | 27 |
| 2.5.1.2 Message Digest 5 (MD5) | 33 |
| 2.6 Overview of Blind Signature | 34 |
| 2.6.1 Application on Online Voting | 36 |
| 2.7 Blind Signature Scheme | 39 |
| 2.7.1 Blinding the RSA Signature Scheme | 39 |
| 2.7.1.1 Blind Signature Protocol | 40 |
| 2.7.2 Blind Schnorr Digital Signature Scheme | 41 |
| 2.7.2.1 The Original of Schnorr Signature Scheme | 41 |
| 2.7.2.2 Blinding the Original of Schnorr Signature Scheme | 42 |
| 2.8 Programming Language | 43 |
| 2.8.1 Microsoft Visual C ++ | 43 |
| 2.8.2 Java | 44 |
| 2.8.3 Visual Basic | 45 |

| | | |
|-------------------|---------------------------------------|----|
| 2.9 | System Review | 45 |
| 2.9.1 | Online Voting System (OVS) | 46 |
| 2.9.2 | RemoteVote | 48 |
| 2.9.3 | SafeGuardSign and Crypt | 50 |
| 2.9.4 | FileAssurity | 51 |
| 2.9.5 | Verisign Code Signing for Digital IDs | 52 |
| 2.9.6 | E-Lock Prosigner | 53 |
| Chapter 3: | Methodology | 58 |
| 3.1 | Waterfall Model | 58 |
| 3.2 | Information Gathering | 60 |
| Chapter 4: | System Analysis | 62 |
| 4.1 | System Requirements | 62 |
| 4.1.2 | Functional Requirements | 62 |
| 4.1.2.1 | The Signing Process | 62 |
| 4.1.2.2 | The Unblinding Process | 64 |
| 4.1.2.3 | The Verifying Process | 65 |
| 4.1.2.4 | Key Generator Module | 67 |
| 4.1.2.5 | About Module | 67 |
| 4.1.2.6 | User Identity Verification Module | 67 |
| 4.1.3 | Non Functional Requirements | 67 |
| 4.2 | Run Time Requirements | 68 |
| 4.3 | Cryptography | 68 |
| 4.3.1 | Encryption Algorithm | 69 |
| 4.4 | Hash Algorithm | 70 |
| 4.5 | Programming Language | 72 |
| Chapter 5: | System Design | 73 |
| 5.1 | System Architecture | 73 |
| 5.2 | Data Flow Diagram | 73 |
| 5.3 | Interface Design | 75 |
| 5.4 | Interface Flow Chart | 78 |
| Chapter 6: | System Implementation | 81 |
| 6.1 | Introduction | 81 |
| 6.2 | Development Environment | 81 |
| 6.2.1 | Hardware Tools | 81 |
| 6.2.2 | Software Tools | 82 |
| 6.3 | System Development Tools | 82 |
| 6.3.1 | User Interface Development | 82 |
| 6.3.2 | User Authentication Dialog | 83 |
| 6.3.2.1 | Log In Dialog | 83 |
| 6.3.2.2 | Error Message | 83 |
| 6.3.2.3 | Blind Key Input Dialog | 84 |
| 6.3.2.4 | Private Key Input Dialog | 84 |
| 6.3.2.5 | Verifying Key Input Dialog | 85 |

| | | |
|--------------------|---------------------------------------|-----|
| | 6.3.2.6 Warning Message | 86 |
| 6.3.3 | Main Interfaces | 87 |
| | 6.3.3.1 Open / Save File | 89 |
| | 6.3.3.2 Generate Key | 90 |
| | 6.3.3.3 Blinding Task | 92 |
| | 6.3.3.4 Signing Task | 93 |
| | 6.3.3.5 Verifying Task | 94 |
| | 6.3.3.6 Exit | 95 |
| 6.3.2 | Code Development | 95 |
| Chapter 7: | System Testing | 108 |
| 7.1 | Introduction | 108 |
| 7.2 | Type of Testing | 108 |
| Chapter 8 : | System Evaluation | 110 |
| 8.1 | Introduction | 110 |
| 8.2 | System Strength | 110 |
| 8.3 | System Limitation | 111 |
| 8.4 | Recommendation for Future Enhancement | 111 |
| 8.5 | Problem Discussion and Solutions | 112 |
| References | | |
| Bibliography | | |
| Appendix | | |

List of Illustrations

Figures:

| | |
|---|----|
| Figure 2.1 Model of Symmetric Encryption | 12 |
| Figure 2.2 The Structure of the Feistel Cipher | 14 |
| Figure 2.3 The Single Iteration of DES Algorithm | 16 |
| Figure 2.4 The Process of Triple DES | 17 |
| Figure 2.5 Public Key Cryptography for Encryption | 20 |
| Figure 2.6 Public Key Cryptography for Authentication | 20 |
| Figure 2.7 Digital Signature Using RSA Approach | 22 |
| Figure 2.8 Digital Signature Using DSA Approach | 24 |
| Figure 2.9 (a) Message Authentication Using Conventional Encryption | 26 |
| Figure 2.9 (b) Message Authentication Using Public Key Encryption | 26 |
| Figure 2.9 (c) Message Authentication Using Secret Value | 27 |
| Figure 2.10 The processing of a Single 512- bit block | 28 |
| Figure 2.11 Message Digest Generation Using SHA-1 | 29 |
| Figure 2.12 The Elementary SHA Operation | 31 |
| Figure 2.1.3 Circular Left Shift Rotation | 32 |
| Figure 2.14 The Signing Process | 38 |
| Figure 2.15 The verifying process | 39 |
| Figure 2.16 The Ballot of the Online Voting System(OVS) | 46 |
| Figure 2.17 The Administrative Side of Online Voting System (OVS) | 48 |
| Figure 3.1 The Waterfall Model | 60 |
| Figure 5.1 Overall System Architecture | 73 |
| Figure 5.2 Data Flow Diagram | 74 |
| Figure 5.3 The System Main Interface Prototype Design | 76 |
| Figure 5.4 Key Pop Up Menu Prototype Design | 77 |
| Figure 5.5 Signing Process Interface prototype Design | 78 |
| Figure 5.6 The Interface Flow Chart | 80 |
| Figure 6.1 Log In Dialog Box | 83 |
| Figure 6.2 Error Message | 83 |
| Figure 6.3 Blind Key Input Dialog | 84 |
| Figure 6.4 Signing Key Input Dialog | 85 |
| Figure 6.5 Verifying Key Input Dialog | 85 |
| Figure 6.6 Error Message Dialog Box (Public Key and Private Key) | 86 |
| Figure 6.7 Error Message Dialog Box (Multiple Value) | 86 |
| Figure 6.8 Error Message Dialog Box (Random Key) | 86 |
| Figure 6.9 Error Message Invalid Signature | 87 |
| Figure 6.10 Informing Message Valid Signature | 87 |
| Figure 6.11 Main Interface | 88 |
| Figure 6.12 Open File Task | 89 |
| Figure 6.13 Save Task | 90 |
| Figure 6.14 Key Generator | 91 |
| Figure 6.15 Blinding Process | 92 |
| Figure 6.16 Signing Process | 93 |
| Figure 6.17 Verifying Process | 94 |
| Figure 6.18 Closing Program | 95 |

Tables:

| | |
|---|----|
| Table 2.1 Summarization of the Using Algorithm on the Existing System | 56 |
| Table 4.1 The Differences and Similarities between SHA-1 and MD5 | 70 |
| Table 4.2 Performance of MD5 and SHA-1 Algorithm on 850 MHz Celeron | 71 |

University of Malaya

Chapter 1 Introduction

1.1 Aim

This thesis introduces the notion of blind signatures and provides a construction which enables us to realize this notion. Blind signature is capable to protect the integrity and authenticity of the data by enables the signer to sign the document without knowing the contents. Therefore, through this research, the author should be competent to understand and implement the knowledge of blind signature through the software that will be develop at the end of this research.

1.2 Problem and Motivation

Computerized transactions of all kinds are becoming ever more pervasive, nowadays. Because of this phenomenon, security is most important thing that simply to be done since the amounts of money are involved in every task of the scheme. Therefore, there should be a technique that can avoid the possibility of fraud during the transaction. As a result, the first approach of this crisis which is digital signature is introduced.

Digital signature is the electronic analog of the traditional handwritten signature. This scheme allowed the signer to sign the document using the private key and only parties that have the signer's public key can verify the document to proof that the document is sent by the signer. Although this method is considered as a secured system, but it has no privacy. This is because the parties whoever involved in this virtual communication possibly will track where or what the purpose of the transaction that is being done. Supposed that the purchased of goods using electronic cash is untraceable or the voting is progress without revealing the identity of the voter.

As a consequence, blind signature scheme is proposed to make sure the transaction between two parties is secure and untraceable to protect the individual's privacy.

Thus, because of the insufficiency of digital signature scheme, it motivates the author to make a deep research on blind signature. Unlike the digital signature, blind signature allowed the signer to sign the document without revealing the content.

1.3 Objectives

In order to have a deep understanding on blind signature, the objectives of this research have been made as guidance to the author while producing a good report.

a) Background research on blind signature

Preliminary study on what is exactly the blind signature and how they work is done. Besides, the investigation on the existing encryption and authentication method that being used in blind signature scheme is made to make the understanding of blind signature as clear as possible.

b) Generation of the blind signature ideas

After a deep research on the existing method of encryption and authentication method, the determination on what kind of method that the blind signature should operate is decided.

c) Hardware and software investigation and comparison

The comparison of the performance on the existing hardware and software should be made to help the author on deciding what the specification of hardware and software that the blind signature system should operate.

d) Familiarize with the algorithms of hash function and encryption function

The research on the flow and the architecture of the algorithms of hash and encryption function is totally completed to make sure the implementation of the algorithms is successful.

e) Understanding of data flow in blind signature system

The flow of the system should be fully understood to make sure the progress of the system work as the author anticipates.

f) System design

The system was designed based on the data flow diagram.

1.4 Project Scope

Below are the scope of the blind signature system should operate:

- This system had basic features that blind signatures system required such as blinding the document, unblinding the document, signing and verifying the document.
- Only two parties can be involved each time a communication is being held.
- The system is developed based on the application of online transaction which required the same level of security.

1.5 Limitations

While developing the blind signature system, a few limitations had been discovered which are stated below:

- Although the public key encryption is much more secure than secret key encryption, but the speed on using public key encryption is much slower than secret key encryption.
- Besides, anything changes in a signed document will affect the verification process. System will fail to verify the signature although it caused by a transmission error not the attempt to forge the signature.
- If the third party finds out the private key and the random key, the possibility to forge the signature or to reveal the content of the signature is high.
- If unauthorized third party know how the exact calculation on producing the individual's private key by deriving it from public key, it is possible to the attacker to reproduce a new private key.

Chapter 2 Literature Review

2.1 The background of Cryptography

¹Cryptography existed since 4000 years ago and the usage of this method become more vital day by day as a consequence to tremendous growth of internet. Thus, the ²cryptanalyst has struggles to bring up the new technique of cryptography that significance to the rapid development on computer technology. Cryptography can described as a complex mathematical technique of encoding a ³plaintext to ⁴ciphertext to avoid any unauthorized parties to read or alter the text. Modern cryptography concerns itself with the following four objectives:

- 1) Confidentiality - the information cannot be understood by anyone for whom it was unintended
- 2) Integrity - the information cannot be altered in storage or transit between sender and intended receiver without the alteration being detected.
- 3) Non-repudiation - the creator/sender of the information cannot deny at a later stage his or her intentions in the creation or transmission of the information.
- 4) Authentication - the sender and receiver can confirm each others identity and the origin/destination of the information.

¹ Cryptography is the art of concealing information using encryption [Eric Maiwald, 2000]

² An individual who use cryptanalysis to identify and use weaknesses in cryptographic algorithms. [Eric Maiwald, 2000]

³ Plaintext is the original message or data that is fed into the algorithm as input [William Stallings, 2000]

⁴ Ciphertext is the scrambled message produced as output. It depends on the plaintext and the secret key. For a given message, two different keys will produce two different ciphertext. [William Stallings, 2000]

[http://searchsecurity.techtarget.com/sDefinition/0,,sid14_gci214431,00.html,
September 2004]

Below are a few techniques for transforming the plaintext to ciphertext:

a) Substitution

i) Polyalphabetic

Periodic

Non-Interrelated Alphabets

Interrelated Alphabets

Pseudorandom key

Non periodic

Non random key, random key

ii) Polygraphic

Digraphic, Algebraic

iii) Monoalphabetic

Standard, Mixed Alphabet, Homomorphic, Incomplete Mixed Alphabet,
Multiplex, Double

iv) Fractionating

Bifid, Trifid, Fractionated Morse, Morbit

b) Transposition

i) Geometrical – Rail fence, Route, Grille

ii) Columnar

Complete – Cadenus, Nihilist

Incomplete – Myskowski, Amsco

iii) Double – U.S Army Transposition Cipher

There are, in general, three types of modern cryptography algorithm which are secret key (or symmetric) algorithms, public-key (or asymmetric) algorithms, and hash functions algorithms. Besides, there are also three types of secret key cryptographic primitives: additive stream cipher, cryptographic hash functions and block ciphers. These six types of cryptographic will be explained through out this chapter. But the author will concentrate on the modern cryptography.

2.2 Cryptographic Primitives

While developing any application (software), security is the most important thing that needs to be done. Therefore, nowadays numerous applications use implementation of cryptographic algorithm to provide a security that resistance against attacks and at a low cost. Besides, the implementation of the algorithm also must not reduce the performance

of the application. One of the advantages of primitives cryptography is that it is usually much faster than public-key cryptography. This is because primitives cryptography only used a single key (secret key) to encrypt and decrypt the message. But the difficulty with secret key cryptosystems is sharing a key between the sender and receiver without anyone else compromising it. In a system supporting a large number of users the key management problems can become very severe. Three types of cryptography primitives are discussed in this section. Block ciphers are used to encrypt data. If block ciphers is not fast enough, additive stream ciphers are used as an alternative. Besides, in order to ensure the integrity of data cryptographic hash functions are used.

2.2.1 Block Ciphers

A block cipher is defined as a set of Boolean permutation operating on n -bit vectors. This set contains a Boolean permutation for each value of a key. In other words, it transforms a fixed-length block of plaintext into a block of ciphertext of the same length by using a secret key. A block cipher usually consists of several operations which are:

- Electronic Codebook (ECB) mode is the simplest, most obvious application: the secret key is used to encrypt the plaintext block to form a ciphertext block. Two identical plaintext blocks, then, will always generate the same ciphertext block. Although this is the most common mode of block ciphers, it is susceptible to a variety of brute-force attacks.
- Cipher Block Chaining (CBC) mode adds a feedback mechanism to the encryption scheme. In CBC, the plaintext is exclusively-ORed (XORed) with the

previous ciphertext block prior to encryption. In this mode, two identical blocks of plaintext never encrypt to the same ciphertext.

- Cipher Feedback (CFB) mode is a block cipher implementation as a self-synchronizing stream cipher. CFB mode allows data to be encrypted in units smaller than the block size, which might be useful in some applications such as encrypting
 - interactive terminal input. If we were using 1-byte CFB mode, for example, each incoming character is placed into a shift register the same size as the block, encrypted, and the block transmitted. At the receiving side, the ciphertext is decrypted and the extra bits in the block (i.e., everything above and beyond the one byte) are discarded.
 - Output Feedback (OFB) mode is a block cipher implementation conceptually similar to a synchronous stream cipher. OFB prevents the same plaintext block from generating the same ciphertext block by using an internal feedback mechanism that is independent of both the plaintext and ciphertext bitstreams.
- [<http://www.garykessler.net/library/crypto.html#fig01>, September 2004]

To allow efficient implementation, block ciphers apply the same Boolean transformation several times on a plaintext. Most good block ciphers transform the secret key into a number of sub keys and the data is encrypted by a process that has several rounds (iterations) each round using a different sub key. The set of sub keys is known as the

key schedule. Block cipher can also be used to construct other primitives such as hash functions, and ⁵MACs.

2.2.2 Additive stream ciphers

Stream ciphers encrypt individual characters, which are usually bits, of a plaintext one at a time. Stream ciphers are typically much faster than block ciphers that generate a key stream (a sequence of bits or bytes used as a key). The plaintext is combined with the key stream, usually with the XOR operation. There are two techniques of stream ciphers which are Synchronous stream ciphers and Asynchronous stream ciphers. Synchronous stream ciphers generate a keystream independently of the plaintext message and of the ciphertext. Sender and receiver must be synchronized; they must use the same key and operate at the same state within that key. Asynchronous stream ciphers are stream ciphers in which the keystream is generated as a function of the key and a fixed number of previous ciphertext bits.

2.2.3 Cryptographic Hash Functions

Cryptographic hash functions compress an input of arbitrary length to an output of fixed length which is called the hash value. They satisfy the following properties:

- **Preimage resistance:** For any given code h , it is computationally infeasible to find x such that $H(x) = h$.
- **Collision resistance:** For any given block x , it is computationally infeasible to find $y \neq x$ with $H(y) = H(x)$.

⁵ Message authentication code is a small block of data generated by a secret key, which will be appended to the message.

- 2nd preimage resistance: It is computationally infeasible to find any pair (x,y) such that $H(x) = H(y)$.
- H can be applied to a block of data of any size
- H produces a fixed-length output.
- $H(x)$ is relatively easy to compute for any given x, making both hardware and software implementation practical. [William Stallings, 2000]

2.3 Secret key Cryptography

Secret key cryptography is also called secret key or symmetric encryption is a method that used a single key to encrypt and decrypt the text. As shown in figure 2.1, suppose that A want to send a confidential message (P) to B. A first need to encrypt the message by using a single key (K) and encryption function (E). Then, the resulting ciphertext,

$C = E_K(P)$ is send to B. B then need to decrypt the message by using the same key and the decrypt function. The key here is produced by the third party who is the key distribution centre (KDC) and during distribution of the key; it must be secured in terms of confidentiality, integrity and authenticity. Symmetric key cryptography has several weaknesses:

- Key distribution is a major problem. Parties must have a secure method of exchanging the secret key before establishing communications with the symmetric key protocol. If a secure electronic channel is not available, an offline key distribution method must often be used.

- Symmetric key cryptography does not implement nonrepudiation. Because any communicating party can encrypt and decrypt messages with the shared secret key, there is no way to tell where a given message originated.
- The algorithm is not scalable. It is extremely difficult for large groups to communicate using symmetric key cryptography. Secure private communication between individuals in the group could be achieved only if each possible combination of users shared a private key.
- Keys must be regenerated often. Each time a participant leaves the group, all keys that involved that participant must be discarded.

[Ed Tittel, Mike Chapple and James Michael Stewart, 2003]

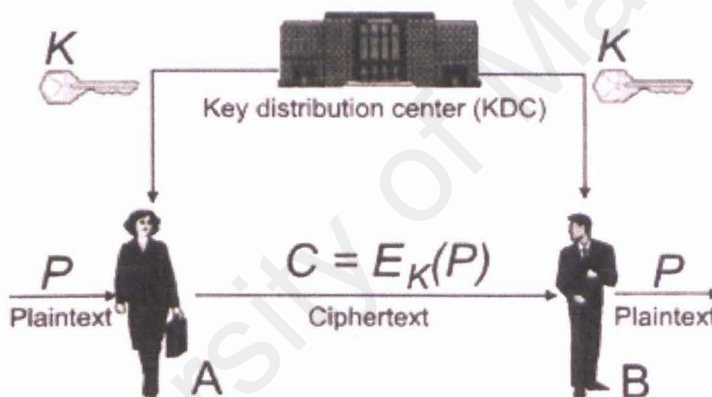


Figure 2.1 Model of Symmetric Encryption.

2.3.1 Feistel Cipher Structure

Feistel cipher structure is one of the modern block ciphers which are devised by Horst Feistel of IBM on 1973. Feistel ciphers are a special class of iterated block ciphers where the ciphertext is calculated from the plaintext by repeated application of the same transformation or round function. There are two main things that to be considered:

- Fast software encryption or decryption- encryption that is embedded in applications or utility functions in order to avoid a hardware implementation will knock down a speed execution.
- Ease of analysis- Cryptanalytic vulnerabilities will be discovered easier if an algorithm can be explained concisely. Therefore, a higher level of assurance as to increase an algorithm's strength can be developed. [William Stallings, 2000]

The structure of the fiestel cipher is showed in figure 2.2. This modern block cipher is usually based on two different types of designs. The plaintext is split into a rightmost w -bits (R) and a leftmost w -bits (L). These two blocks is processed through a few rounds n . Then, round- i (L_i and R_i) will be the input for round- $i+1$, as well as subkeys, K_{i+1} . A substitution is performed on the L data by applying a round function F to the R data and then taking the XOR of the output of that function and the L data. The F function is parameterized by the round subkey K_i . After the substitution, a permutation is performed (interchange between L and R data). As a result, this architecture only depends on the design of block size, key size, number of rounds, subkey generation algorithm and round function.

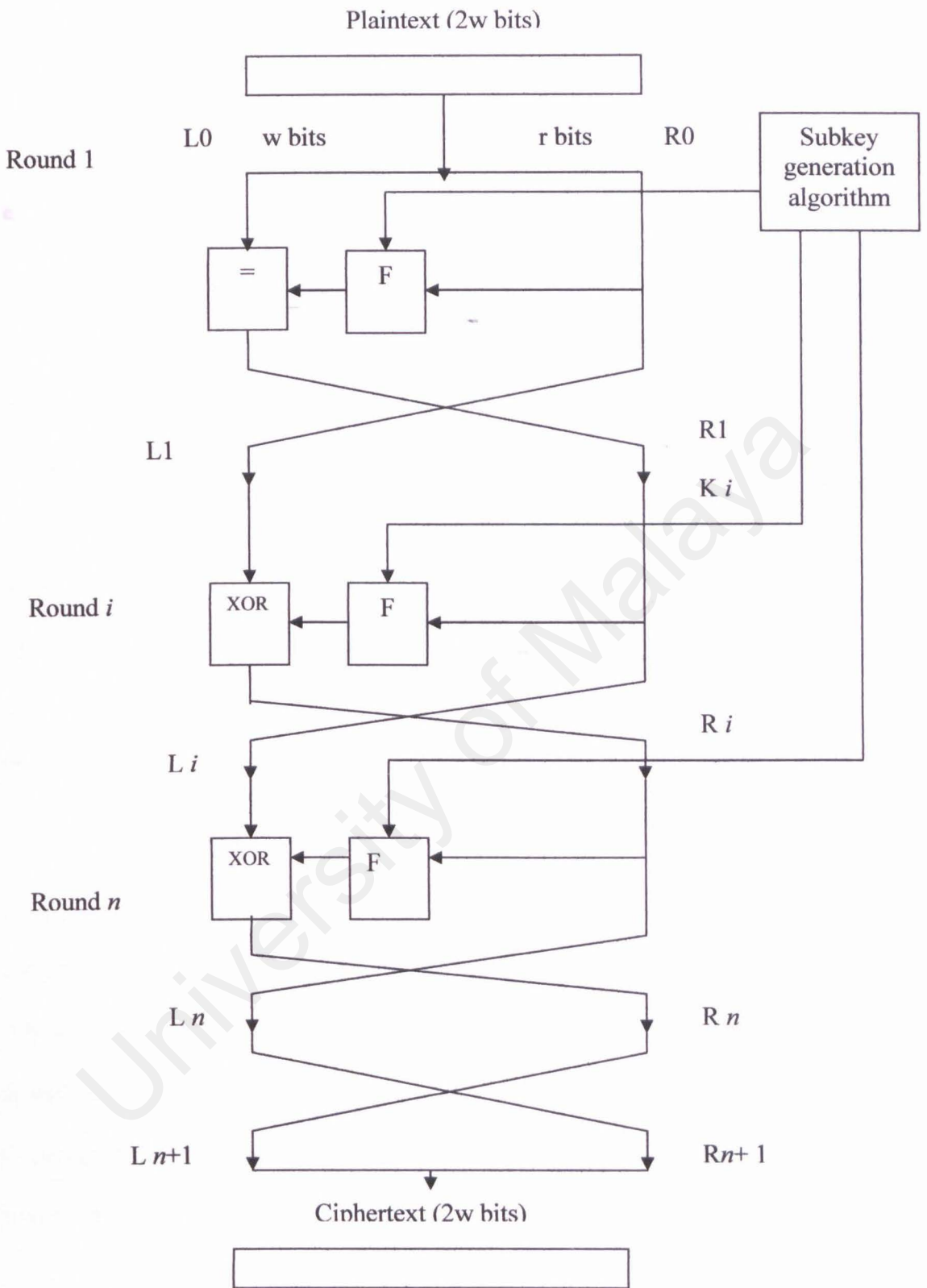


Figure 2.2 The structure of the feistel cipher

2.3.2 Symmetric Encryption Algorithm

2.3.2.1 Data Encryption Standard (DES)

Data Encryption Standard that was developed by IBM on 1977 is a method to encrypt and decrypt the data by using a single key. DES applies 56-bits of a key to each 64-bits block of data. Then it will process in several modes and involves in 16 rounds of encryption. Each round uses ⁶permutation and ⁷substitution operations, and each uses a different 48-bit subkey that was generated from the original 56-bit key. Figure 2.3 shows a single iteration of DES algorithm. [William Stallings, 2000]. Based of the figure, 64-bits block of data (permuted input) is divided into two portions, which are called leftmost (L) and rightmost (R). Each of the iterations will process the permuted input by applying the processing function that can be concluded as below:

$$L_i = R_{i-1}$$

$$R_i = L_{i-1} \text{ XOR } F(R_{i-1}, K_i)$$

The 56-bits key is also divided into two parts which are 28 bits for C_0 and another 28 bits for D_0 . Then both of these parts will go through a circular left shift or rotation of 1 or 2 bits, during each of the iteration. The result of this process which is 48 bits output, then will be the input of the of the permutation function, $F(R_{i-1}, K_i)$. Function F involves both permutation and substitutions operations. The substitution boxes is represented as S-boxes which will maps each combination of 48 input bits into 32 bits pattern.

⁶ Permutation is an operation that jumbles up the bits in a block into new positions. [Mohan Atreya, 2003]

⁷ Substitution is an operation that substitutes a new group of bits for each group of bits output by an initial permutation. [Mohan Atreya, 2003]

Although DES is a very secure algorithm a few years back, but DES is not widely implemented nowadays because DES can be easily break with the help of advance technology that available today. Therefore, Triple Data Encryption Standard (3DES) is introduced. With Triple DES, it can increase the length of the key through the process which called encrypt-decrypt-encrypt. Figure 2.4 shows that the process of Triple DES.

[Suranjan Choudhury, Kartik Bhatnagar, Wasim Haque and NIIT, 2002]

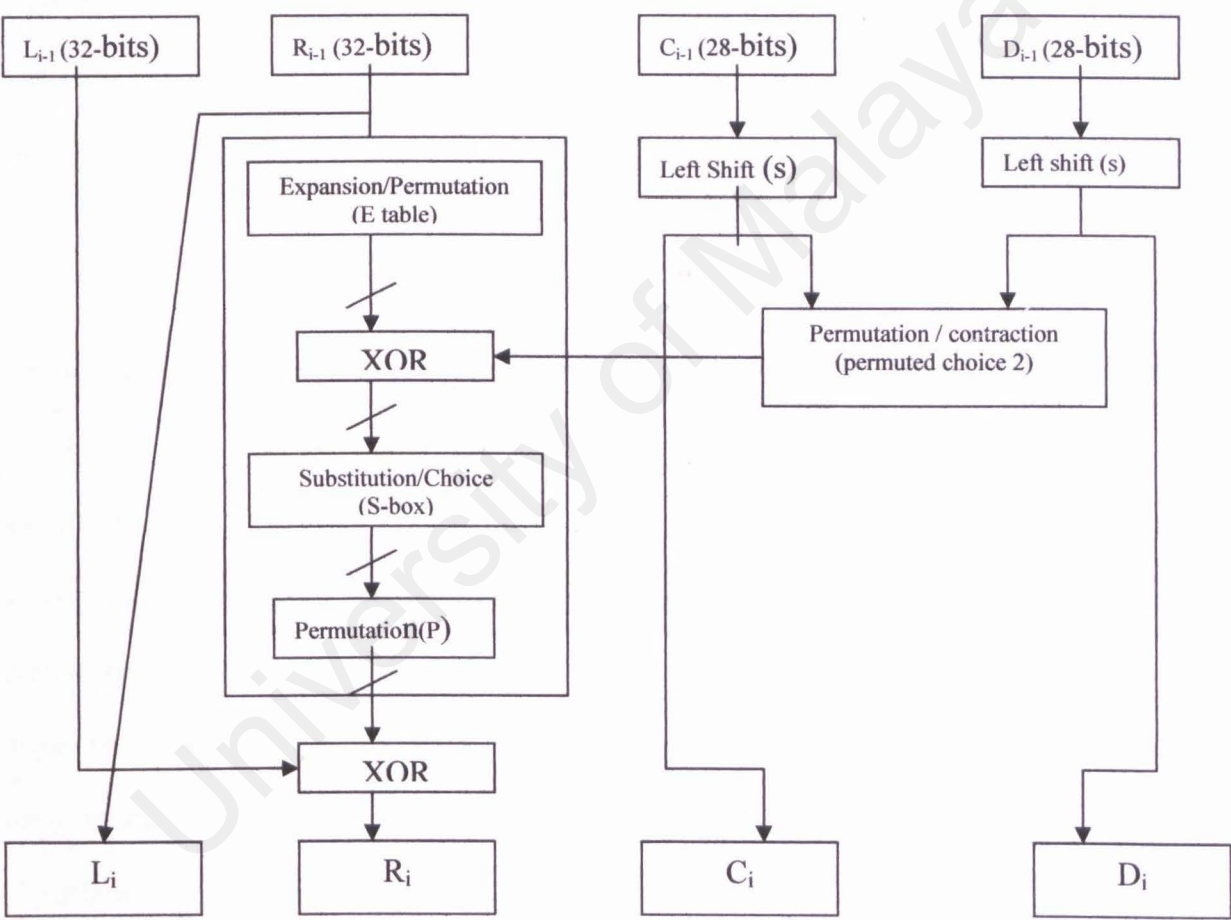


Figure 2.3: The single iteration of DES algorithm

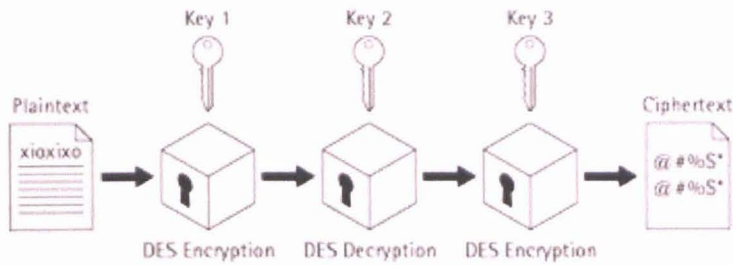


Figure 2.4: The process of Triple DES

Based on the figure, it first encrypts the plaintext using 56-bits key. Then, the ciphertext is decrypted by using a different key. During the decryption it will produce some garbage. Finally, the garbage is encrypted by using the first key. This algorithm is three times slower than DES but it can be much more secure if it used properly.

2.3.2.2 International Data Encryption Algorithm (IDEA)

International Data Encryption Algorithm is developed by James Massey and Xuejia Lai which is patented for Swiss ETH University. This algorithm is considered as a secured algorithm because it used 128-bits key length longer than DES and Triple DES algorithm. It runs eight operations on each block and each round involves three different operations; XOR, addition and multiplication. In addition, the subkey generation algorithm uses circular shifts in a complex way as to generate a total of 6 subkeys for each round. Nevertheless, this algorithm is not popular as DES and Triple DES algorithm because firstly, IDEA is slower than DES but is faster from Triple DES algorithm. Secondly, it cannot be commercially without license and finally, it has not been declared as a federal standard.

2.3.2.3 Blowfish

Blowfish is designed by Bruce Schneier in 1993 as another alternative to DES and IDEA algorithm. It takes a variable length key ranging from a relatively insecure 32 bits to an extremely strong 448 bits. Obviously, the longer key will affect the time of decryption or encryption. However, time trials proof that blowfish algorithm is faster than both DES and IDEA algorithm. The most important element in Blowfish algorithm is Fiestel network. Fiestel uses dynamic S-boxes, which generated as a function of the key, and XOR function and binary addition. It will process in several modes and involves in 16 rounds of encryption. Each round uses permutation and substitution operations, and each uses a different subkey that was generated from the original key. A total of 521 executions of the blowfish encryption algorithm are required to produce the subkeys and S-boxes. Blowfish algorithm is unpatented and license-free, and is available free for all uses.

2.4 Public Key Cryptography

Public key cryptography is also called public key or asymmetric encryption is another method of cryptography that is totally different from secret key cryptography. Public key cryptography is developed by Whitfield Diffie and Martin Hellman in 1976 as conjunction with insecurity of secret key algorithm. As we can see in secret key algorithm, both the sender and the receiver need to agree on the secret key without anyone else finding out. However, opposite with secret key algorithm, public key algorithm allowed both sender and receiver have two different keys. Once the sender want to send the message to the intended receiver, the sender will encrypt the message

using a public key and the receiver will decrypt the message using a private key which is in the sole possession of the intended recipient. With this system, the private key is kept secret whereas the public key can be given to anyone. A user can have their own key pair generator by using good software in order to get full control over the security of their private key. Furthermore, public-key cryptography can be used not only for encryption, but also for authentication (digital signatures). Although the public key cryptography is much more secure but it is slower than secret key cryptography. The asymmetric key set has the following unique characteristics:

- The relationship between the private and public key is such that any cryptographic operation that is performed using one key can only be reversed by the other. Thus a message encrypted using the public key component of the asymmetric key-pair can only be decrypted by the private key of the very same key-pair.
- Unlike symmetric key cryptography, this technique does not require that the sender or receiver exchange any secret information as part of the transaction.

[Nadir Gulzar and Kartik Ganeshan, 2003]

Figure 2.5 and 2.6 below shows the application of public key encryption which are message encryption and message authentication.

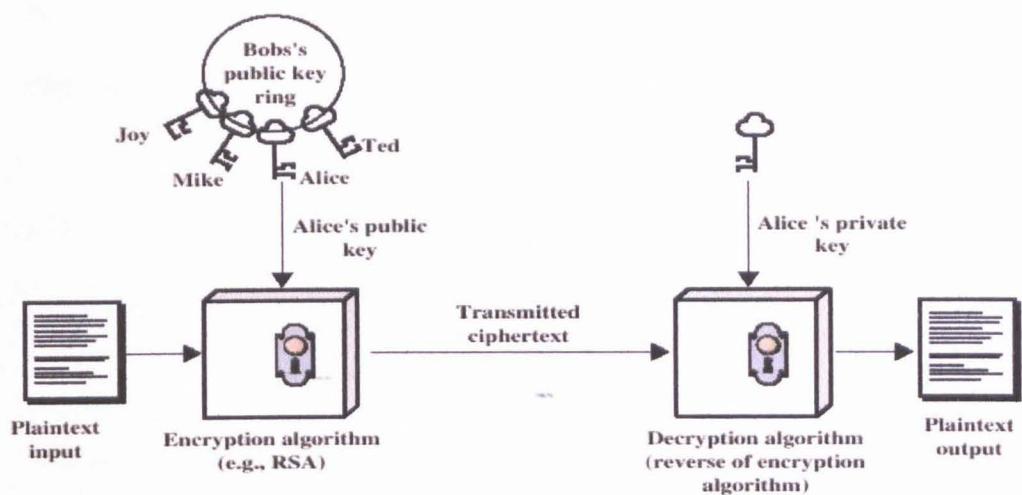


Figure 2.5 Public Key Cryptography for Encryption

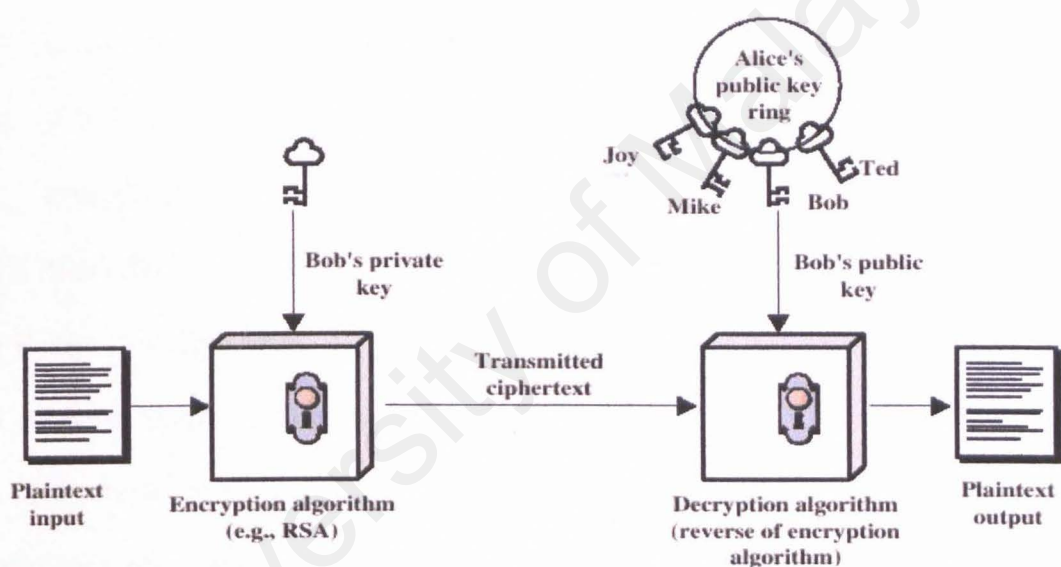


Figure 2.6 Public Key Cryptography for Authentication

There are several conditions had been laid out that the public key encryption algorithm must fulfill:

1. It is computationally easy for a party B to generate a pair (public key K_{Ub} , private K_{Rb}).

2. It is computationally easy for a sender A, knowing the public key and the message to be encrypted, M to generate the corresponding ciphertext:

$$C = EK_{Ub}(M)$$

3. It is computationally easy for a receiver B to decrypt the resulting ciphertext using the private key to recover the original message:

$$M = DK_{Rb}(C) = DK_{Rb}[EK_{Ub}(M)]$$

4. It is computationally infeasible for an opponent, knowing the public key, K_{Ub} , to determine the private key, K_{Rb}
5. It is computationally infeasible for an opponent, knowing the public key, K_{Ub} , and a ciphertext, C, to recover the original message, M.
6. Either of the two related keys can be used for encryption, with the other used for decryption.

[William Stallings, 2000]

2.4.1 Public Key Algorithm

2.4.1.1 Rivest, Shamir, Adleman (RSA)

RSA is developed in 1977 by Ron Rivest, Adi Shamir, and Leonard Adleman. RSA is an internet encryption authentication system that uses an algorithm. Moreover, RSA is a block cipher in which the plaintext and ciphertext are integers between 0 and $n-1$ for some n . [William Stallings, 2000]. Briefly, RSA algorithm involves multiplying two large ⁸prime's number and then a set of two numbers that constitutes the public key and another set that is the private key are derived.

⁸ A prime number is a number divisible only by that number and 1.

Let $n = pq$ where p and q are two large primes and let e be chosen such that $(e, \Phi(n)) = 1$ and $\Phi(n) = (p-1)(q-1)$. Moreover let d be such that $de = 1 \pmod{\Phi(n)}$. Assume that the signer's public key is (n, e) and the private key is (p, q, d) and finally let H be the collision resistant hash function.

For given a message m , is a valid RSA signature if:

$$C = H(m)e \pmod n$$

And the signature is easily verified by using a private key:

$$M = H(m)d \pmod n$$

It is simply to verify that the signature is valid or not by comparing the equality of these two M and m . Figure 2.7 shows the overall process of generating digital signature using RSA.

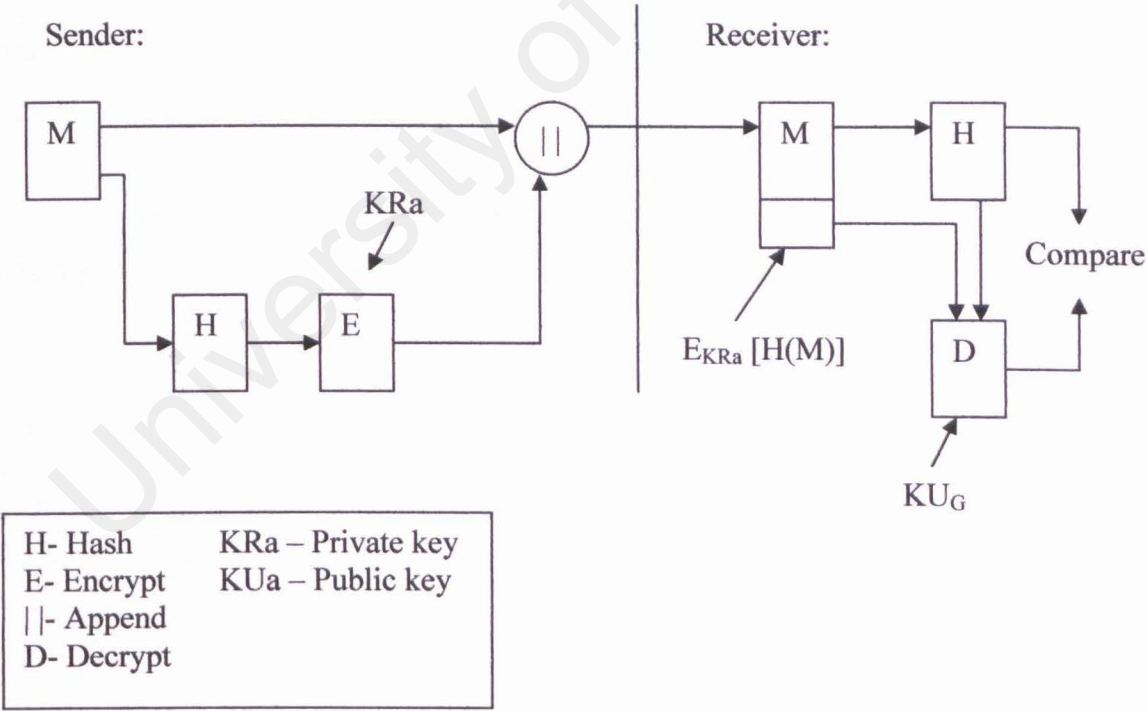


Figure 2.7 Digital Signature using RSA Approach

2.4.1.2 Digital Signature Algorithm (DSA)

Digital Signature Algorithm is public key system for generating digital signature that was designed in 1994 and the use of DSA for digital signature was specified under Digital Signature Standard that is NIST standard (FIPS186). DSA has a more complex architecture and it provides solely the function of digital signature.

Let p be a prime number where $2L-1 < p < 2L$ for $512 \leq L \leq 1024$ and L is a multiple of 64. Moreover, let q be a prime divisor of $(p-1)$, where $2159 < q < 2160$. Then, let $g = h^{(p-1)/q} \bmod p$, where h is any integer with $1 < h < (p-1)$ such that $h^{(p-1)/q} \bmod p > 1$.

Assume that x is a user's public key where x is random or pseudorandom integer with $0 < x < q$ and y is a user's private key where $y = gx \bmod p$.

Besides, let K be a user's per message secret number where K is a random or pseudorandom integer with $0 < K < q$.

The signature (r, s) is done by using:

$$r = (gx \bmod p) \bmod q$$
$$s = [K^{-1} (H(M) + xr)] \bmod q$$

The signature is verified by using:

$$w = (s')^{-1} \bmod q$$
$$U1 = [H(M') w] \bmod q$$
$$U2 = (r') w \bmod q$$
$$v = [(g^{U1} y^{U2}) \bmod p] \bmod q$$
$$\text{TEST: } v = r',$$

where M is a message to be signed, $H(M)$ is a collision resistant hash function and M' , r' , s' is a received version of M , r , s . Figure 2.9 shows the overall process of generating digital signature using DSA.

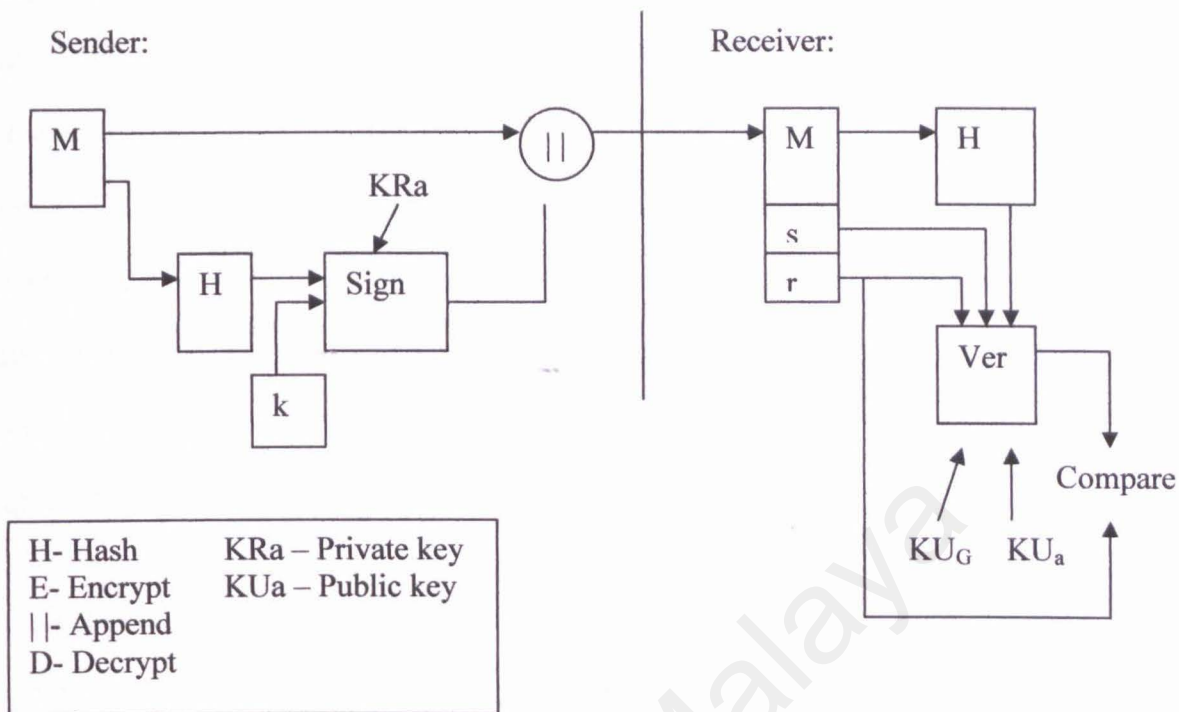


Figure 2.8 Digital Signature using DSA approach

2.5 Hash Functions

A hash function is a function that transforms a variable size input into an output which is a fixed size string where it is called the hash value. The basic requirements for a cryptographic hash function are:

- The input can be of any length,
- The output has a fixed length,
- $H(x)$ is relatively easy to compute for any given x ,
- $H(x)$ is one-way,
- $H(x)$ is collision-free.

[<http://www.x5.net/faqs/crypto/q94.html>, 17 September 2004]

Hashing algorithm is used in message authentication. The two important aspects in message authentication are to verify that the contents of the message have not been altered and that the source is authentic. Message authentication does not rely on encryption. In all of the hashing algorithms, an authentication tag is generated and appended to each message for transmission. Thus, the message itself is not encrypted and can be read at the destination independent of the authentication at the destination. In this context, message confidentiality is not provided. Therefore, message authentication without confidentiality is preferable in some conditions:

- There are a number of applications in which the same message is broadcast to number of destinations (for example, notification to users that the network is now unavailable). It is cheaper and more reliable to have only one destination responsible for monitoring authentication. Thus, the message must be broadcast in plaintext with an associated message authentication tag. The responsible system performs authentication. If a violation occurs, the other destination system are alerted by a general alarm
- An exchange in which one side has a heavy load and cannot afford the time to decrypt all incoming messages. Authentication is carried out on a selective basis, and messages are chosen at random for checking.
- Authentication of a computer program in plaintext is an attractive service. The computer program can be executed without having decrypt it every time authentication tag were attached to the program, it could be checked whenever assurance is required of the integrity of the program. [William Stallings, 2000]

2.5.1 One Way Hash Function

One way hash functions is a mathematical function which takes a variable-length input string and converts it into a fixed-length binary sequence and it is designed in such a way that is hard to reverse the process. In message authentication, the message is sent along with message digest (hash value). There are three ways in which the message can be authenticated. Figure 2.9 (a) shows the message digest can be encrypted using conventional encryption, Figure 2.9 (b) shows the message digest can also be encrypted using public key encryption and Figure 2.9 (c) shows encryption is done using secret value.

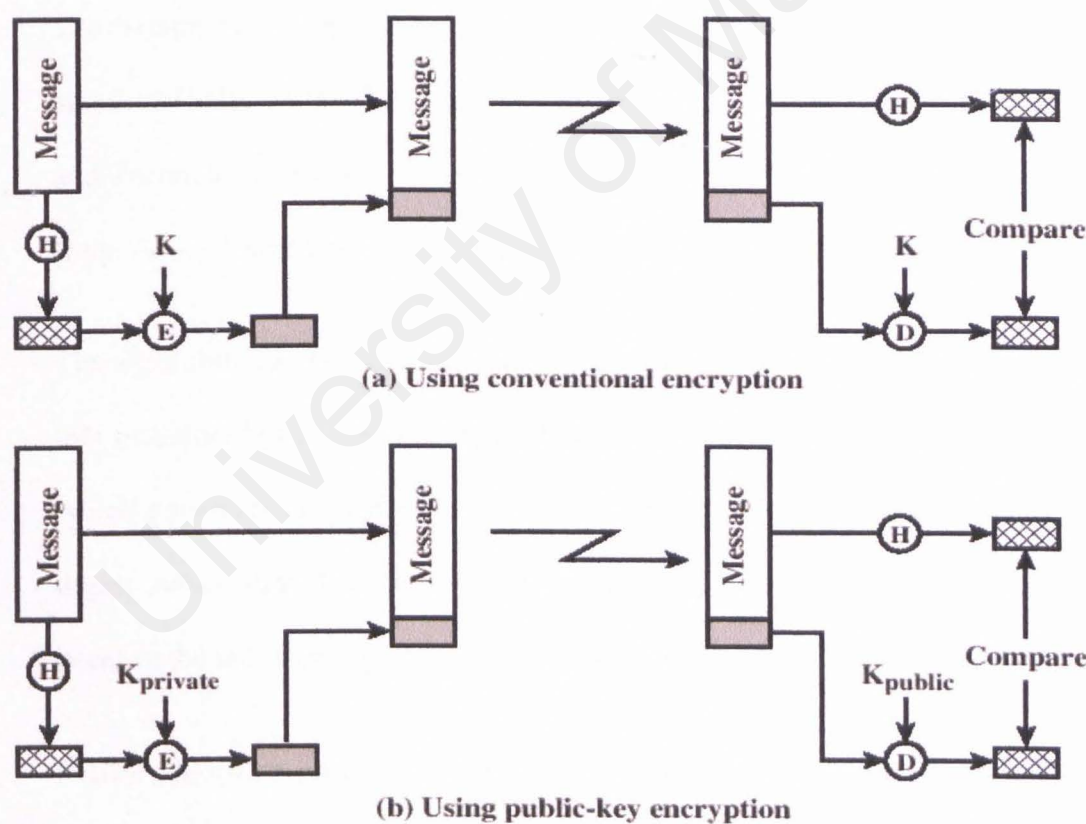


Figure 2.9 (a) and 2.9 (b) Message Authentication using Conventional and Public Key Encryption

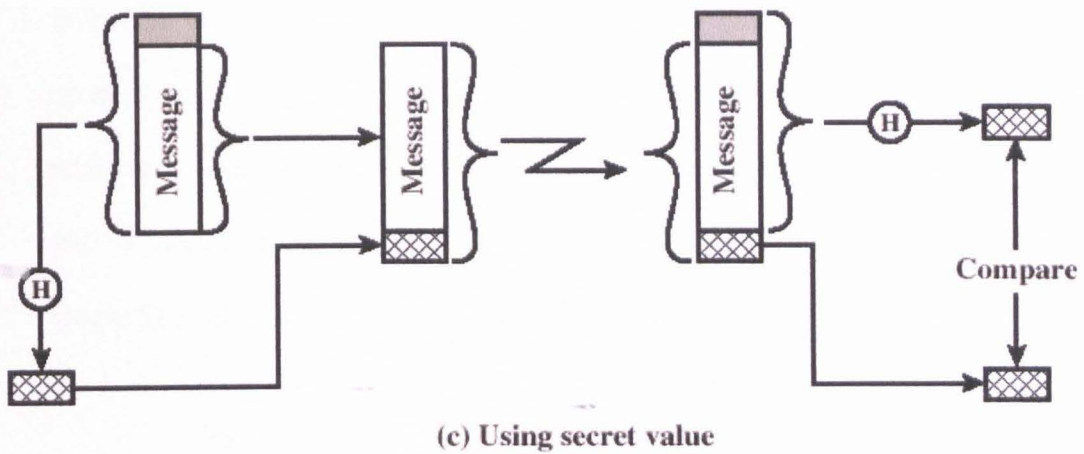


Figure 2.9 (c) Message Authentication using Secret Value

2.5.1.1 Secure Hash Algorithm (SHA-1)

The Secure Hash Algorithm (SHA), the algorithm specified in the secure hash standard (SHS, FIPS 180), was developed by the National Institute of Standard and Technology (NIST). SHA-1 is a revision to SHA that was published in 1994. [http://www.rsasecurity.com/rsalabs/node.asp?id=2252, 17 September 2004].

The algorithm takes a message of less than 2^{64} bits in length and produces a 160 bits message digest. The message digest can be the input of signature algorithm which generates or verifies the signature for the message. Signing the message digest rather than the message often improves the efficiency of the process because the message digest is usually much smaller in size than the message.

SHA-1 algorithm process message in 512-bit (16 word) blocks with compression function, that consists of 4 rounds of processing of 20 steps each. Each round uses different primitive logical function, f as shown in figure. Each round takes as

input the current 512-bit block being processed (Y_q) and value in buffer and updates the contents of the buffer. Each subsequence round will make use of an additive constant K_t as shown in table respectively. The output of the final round will be added to the input to the first round. Figure 3 show the processing of a single 512-bit block and Figure 3.1 show message digest generation using SHA-1.

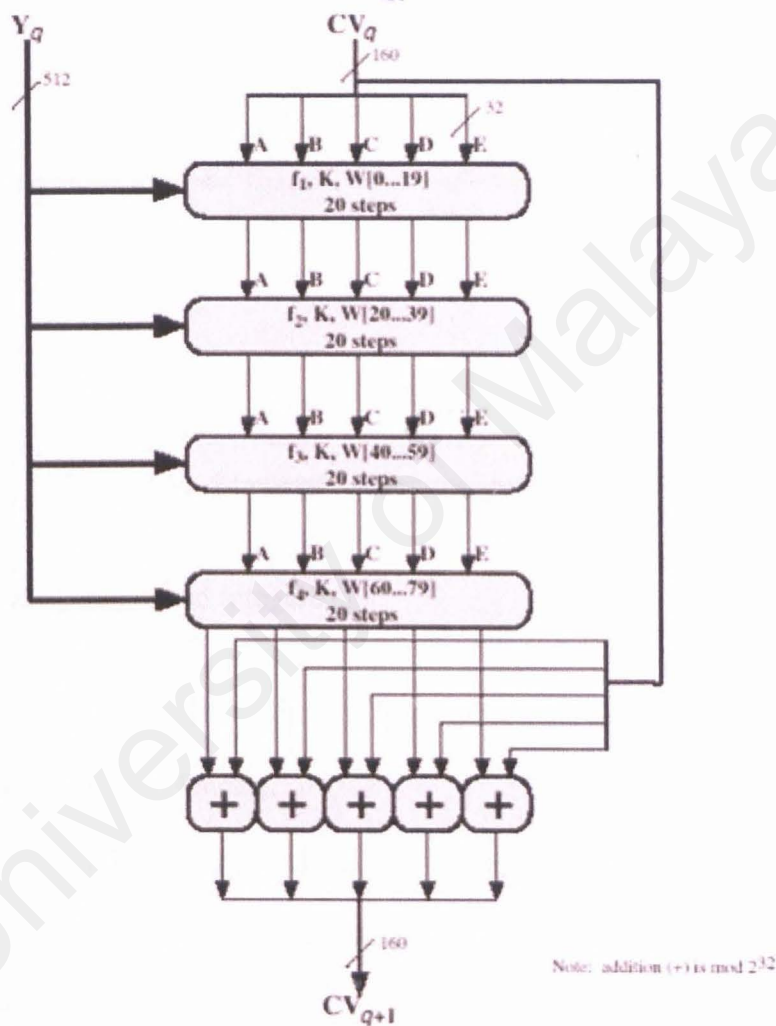


Figure 2.10 The processing of a single 512-bit block

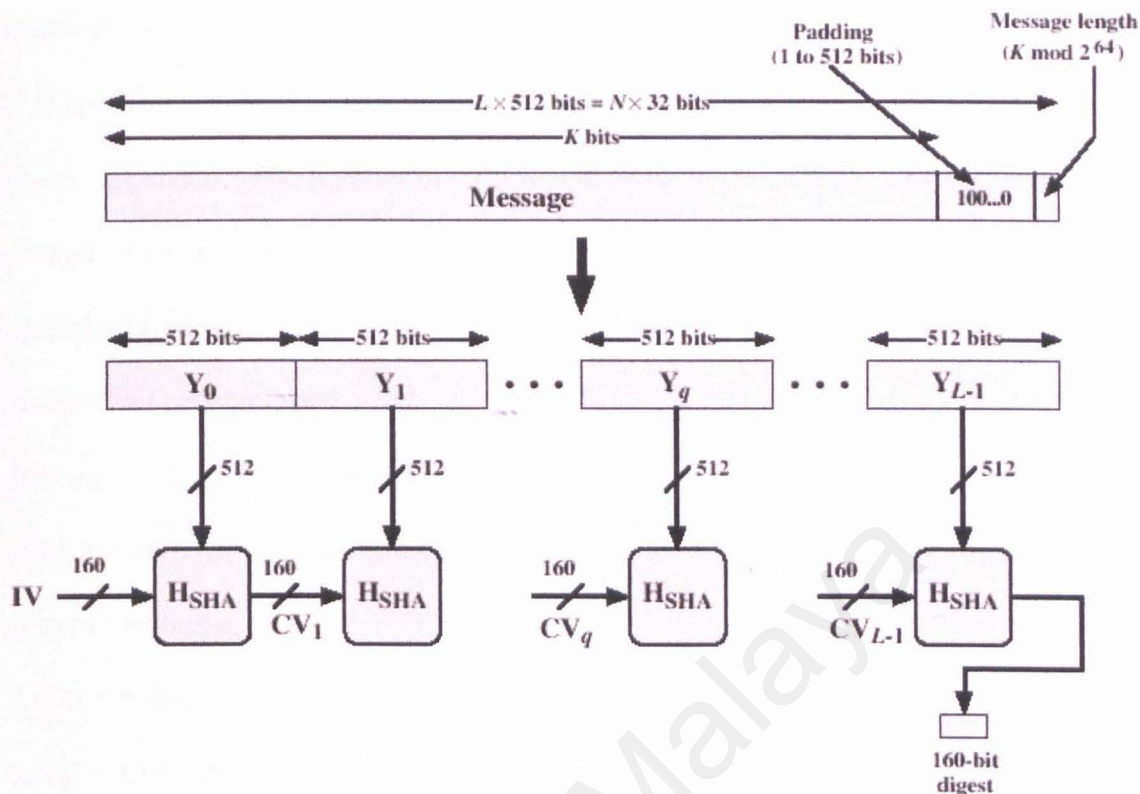


Figure 2.11 Message Digest generation using SHA-1.

Definition of bit string and integers:

- A hex digit is an element of the set $\{0, 1, \dots, 9, A, \dots, F\}$. A hex digit is the representation of a 4-bit string.
- A word equals a 32-bit string which may be represented as a sequence of 8 hex digits. To convert a word to 8 hex digits each 4-bit string is converted to its hex equivalent.
- An integer between 0 and $2^{32} - 1$ inclusive may be represented as a word. The least significant four bits of the integer are represented by the right-most hex digit of the word representation.
- Block = 512-bit string.

The pre processing stage of SHA-1:

- "1" is appended at the end of the original message.
- "0"s are appended. The number of "0"s will depend on the original length of the message. The last 64 bits of the last 512-bit block are reserved for the length of the original message.

Function and constant used:

- Initialize a 160 bit MD buffer (Hi) which is used to hold intermediate and final results of the hash function (message digest). A, B, C, D, E are 32 bit register represent the buffer.

1. $H_0 = 67452301$

2. $H_1 = \text{EFC DAB89}$

3. $H_2 = 98\text{BADCFE}$

4. $H_3 = 10325476$

5. $H_4 = \text{C3D2E1F0}$

- A sequence of logical functions $f(0), f(1), \dots, f(79)$ is used in SHA-1. Each $f(t)$, $0 \leq t \leq 79$, operates on three 32-bit words B, C, D and produces a 32-bit word as output. $f(t; B, C, D)$ is defined as follows: for words B, C, D,

1. $f(t; B, C, D) = (B \text{ AND } C) \text{ OR } ((\text{NOT } B) \text{ AND } D) \quad (0 \leq t \leq 19)$

2. $f(t; B, C, D) = B \text{ XOR } C \text{ XOR } D \quad (20 \leq t \leq 39)$

3. $f(t; B, C, D) = (B \text{ AND } C) \text{ OR } (B \text{ AND } D) \text{ OR } (C \text{ AND } D) \quad (40 \leq t \leq 59)$

4. $f(t; B, C, D) = B \text{ XOR } C \text{ XOR } D \quad (60 \leq t \leq 79).$

- A sequence of constant words $K(0), K(1), \dots, K(79)$ is used in the SHA-1. In hex these are given by:

1. $K(t) = 5A827999$ ($0 \leq t \leq 19$)
2. $K(t) = 6ED9EBA1$ ($20 \leq t \leq 39$)
3. $K(t) = 8F1BBCDC$ ($40 \leq t \leq 59$)
4. $K(t) = CA62C1D6$ ($60 \leq t \leq 79$)

Computing the message digest.

As mentioned above, the words of the first 5-word buffer are labeled A, B, C, D, E, the words of the second 5-word buffer are labeled H0, H1, H2, H3, H4, the words of the 80-word sequence are labeled $f(0), f(1), \dots, f(79)$ and the 16-words blocks $W(1), W(2), \dots, W(15)$ now is processed. Figure 3.2 show the elementary SHA operation. Each round will perform:

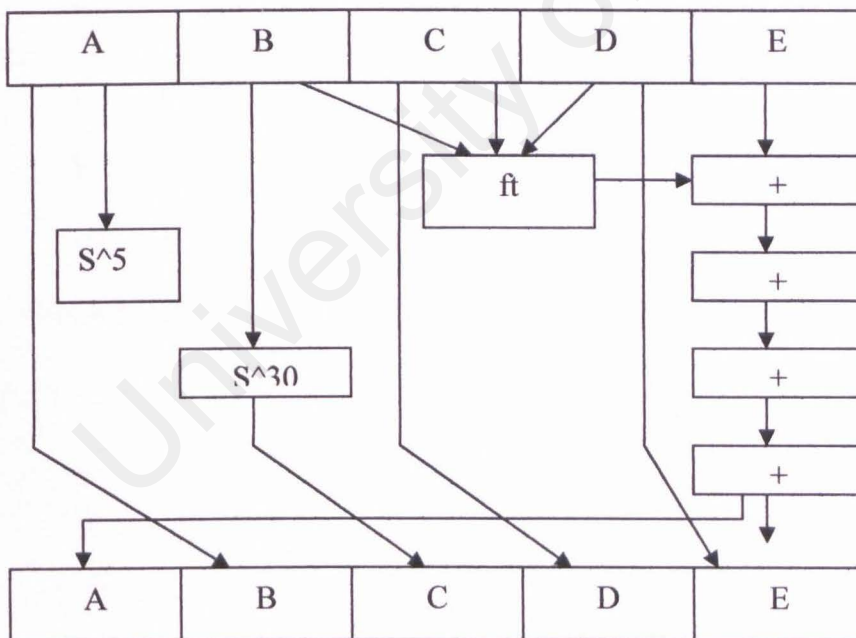


Figure 2.12 The elementary SHA operation

- $A, B, C, D, E \leftarrow (C + f(t, B, C, D) + S^5(A) + W_t + K_t), A, S^{30}(B), C, D$
- $A \leftarrow (C + f(t, B, C, D) + S^5(A) + W_t + K_t)$
- $B \leftarrow A$
- $C \leftarrow S^{30}(B)$
- $D \leftarrow C$
- $E \leftarrow D$

For $t = 0$ to 79 ,

$$TEMP = S^5(A) + f(t, B, C, D) + E + W_t + K_t$$

$$E = D, D = C, C = S^{30}(B), B = A, A = TEMP$$

For $t = 16$ to 79 let $W_t = S^1(W_{t-3} \text{ XOR } W_{t-8} \text{ XOR } W_{t-14} \text{ XOR } W_{t-16})$

Let $A = H_0, B = H_1, C = H_2, D = H_3, E = H_4$

Let $H_0 = H_0 + A$

$$H_1 = H_1 + B$$

$$H_2 = H_2 + C$$

$$H_3 = H_3 + D$$

$$H_4 = H_4 + E$$

S^K = circular left shift rotation of the 32 bit argument by K bits.



Figure 2.13 Circular left shift rotation

2.5.1.2 Message Digest 5

MD5 is a message digest algorithm developed by Rivest that meant for a digital signature application where large message has to 'compressed' in a secure manner before being signed with the private key. This algorithm takes a message of arbitrary length and produces 128 bit message digest. It guessed that the difficulty of coming up with two messages having the same message digest is on the order of 2^{64} operations, and that the difficulty coming up with any message having a given message digest is on the order of 2^{128} operations. The algorithm consists of four distinct rounds, which has a slightly different design from that of MD4, and message digest size, as well as padding requirements, remains the same with SHA-1. Den Boer and Bosselaers have found pseudo-collisions for MD5. More recent work by Dobbertin has extended the techniques used so effectively in the analysis of MD4 to find collisions for the compression function of MD5. While stopping short of providing collisions for the hash function in its entirety this is clearly a significant step. The MD5 differs from MD4 are:

- A fourth round has been added
- Each step now has a unique additive constant.
- The function g in round 2 was changed from $(XY \vee XZ \vee YZ)$ to $(XZ \vee Y \text{ not } (Z))$ to make g less symmetric.
- Each step now adds in the result of the previous step. This promotes a faster "avalanche effect".
- The order in which input words are accessed in rounds 2 and 3 is changed, to make these patterns less like each other.

- The shift amounts in each round have been approximately optimized, to yield a faster “avalanche effect”. The shifts in different rounds are distinct.

[William Stallings, 2003]

2.6 Overview of Blind Signature

Digital signature scheme that proposed by Whitfield Diffie in 1976 that enable people to digitally sign the document during any online transaction to guarantee that the individual sending the message really is who he or she claims to be. This scheme used the cryptography concept where the sender and the receiver need to have a private key and public key to sign and verify the message. Supposed that, the sender who wants to send the message will sign the message using sender's private key and the receiver will verify the message by the sender's public key to make sure the message is utterly from the one who he or she claims to be. This scenario can be described as Alice who wants to pay for a purchase at Bob's shop using her 'digital coin'. In this case 'digital coin' is signed by the bank which using bank's private key. When she pays the purchased to Bob, he will verify the 'digital coin' by using bank's public key as consequences to proof that the coin is valid. Bob then send the 'digital coin' to the bank and the bank will reverify the 'digital coin' to make sure the coin is signed by the bank and belong to Alice. After that, the bank will credit the money to Bob. The bank also will send the slip as a proof that Alice had used the 'digital coin'. So, as we can see here, this approach provides security for all three parties Alice, Bob and the bank. Bob cannot issue that it did not receive the payment; the bank cannot deny that it had credit to Bob, and Alice can neither deny that she had spent the 'digital coin' to Bob nor spend the 'digital coin' twice.

As a conclusion, we can say that the money transaction using digital signature here is secured but it has no privacy. This is because the bank possibility will determine precisely where and when Alice spends her money. Therefore, to enhance this scheme become more reliable blind signature scheme is introduced.

Blind signature approached is introduced by David Chaum that enable people to digitally sign the message without knowing the contents of the message. In other words, the signer did not know when and for whom it signed even though it can verify that signature is indeed valid. According to the first example, supposed that the bank will know the two parties that involved in the transaction and for what purposed the transaction is done. But by using blind signature scheme, the bank would not know what is going on between this two parties and when the event is occurred. Hence, in this scenario Alice who is a spender is retained anonymity in the transaction. The bank will sign the 'digital coin' blindly without knowing to whom the coin is belonged and spent but the bank can verify the coin to admit that the coin is indeed valid. As a result, the event that existed between Alice and Bob is secured and untraceable. At first, when this scheme is introduced many people felt hesitate because how can to sign a document without knowing the content of the document. However, after a deep research is made to this scheme, this approached is utterly accepted. Therefore, here the author will explain in detail how the blind signature scheme work by enlighten through the application of the blind signature scheme. One of the applications that totally used this approached is application to online voting.

2.6.1 Application to Online Voting

There are two parties that involved in online voting system. Which are the voter and the voting checker. When we talk about manual voting system there are two restrictions that we have to consider which are the individual vote is undisclosed to other peoples including the voting checker who is responsible to count the ballot and only once voting is permitted to one person. Therefore, if we want to implement the online voting we must used the blind signature scheme that able to solve these two restrictions. In this scenario we describe Alice is a voter and the other party is Voting Checker Facility. The voting protocol is divided into two phases. First is the registration phase and the second is the voting phase.

a) Registration Phase

Alice will create two ballots which are one for 'yes' and another one for 'no'. We assume that the votes are in the form of 'yes' or 'no'. Both of these ballots consist of the serial number to avoid the people voting more than once and other relevant information about the voter. She then will blindly signed these two ballots and send them to Voting Checker Facility. The Voting Checker Facility will check the serial number in their database to make sure that Alice did not vote before. Then, the Voting Checker Facility will blindly signed the two ballots and send back to Alice.

b) Voting Phase

After receiving her ballots from the Voting Checker Facility, she then unblinds the ballots. She now has two set valid ballots signed by the Voting Checker Facility. Alice picks either 'yes' or 'no' ballots. The selected ballot will encrypt using Voting Checker Facility's public key. Then, she sends in the vote. The Voting Checker Facility decrypts

the ballot, check the database to make sure that Alice did not vote before. Finally, Alice vote has counted and the serial number is record to it's database.

Figure 3.4 and the figure 3.5 below shows the process of signing the blinded document at the sender side and the process of verifying the signature at the receiver side. To blindly sign the document, the document itself must be blinded. The hash of the document must be created first. Then, the hash document will be blind. After that, the blinded hash is encrypted using private key. The encrypted hash is blind signature. The document now is ready to send. The receiver will receive it and verify the signature whether it is indeed valid or not by using the public key.

Sender:

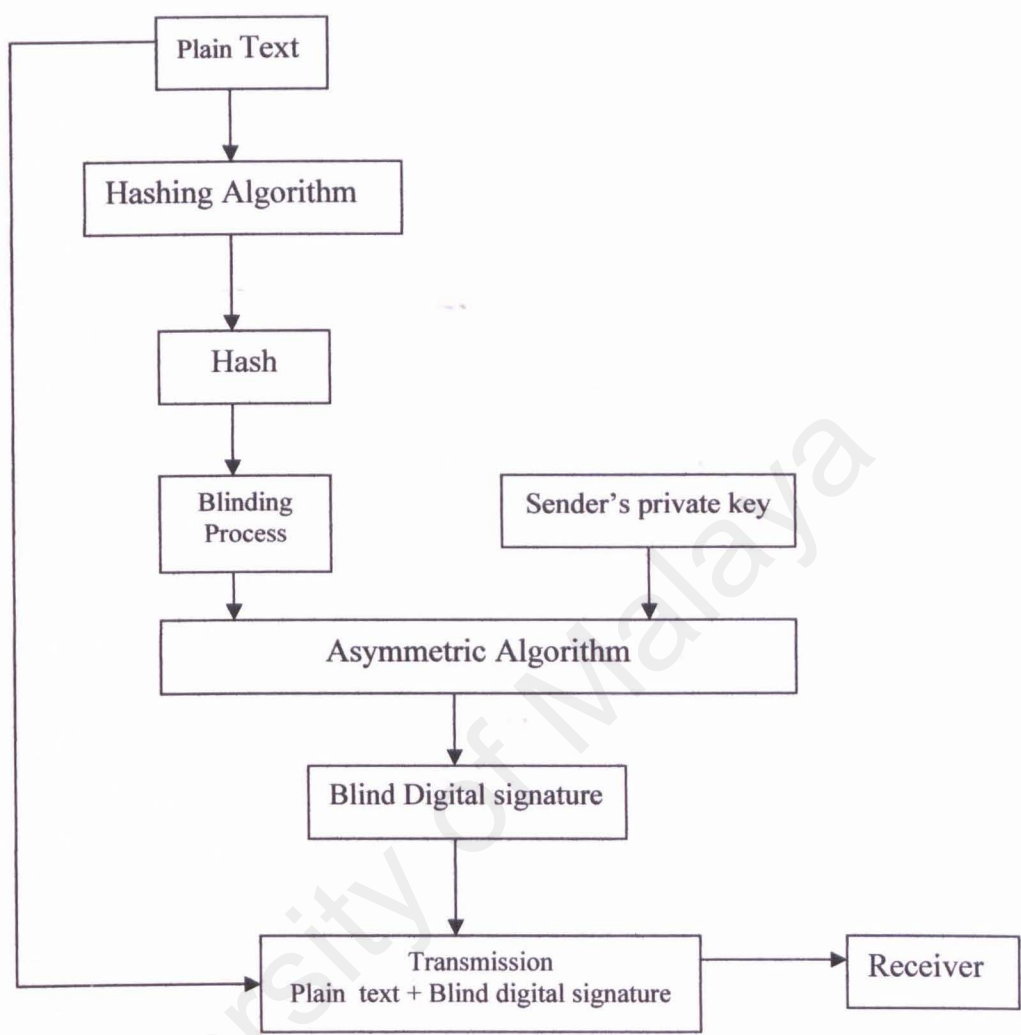


Figure 2.14 Signing process: Sender

Receiver:

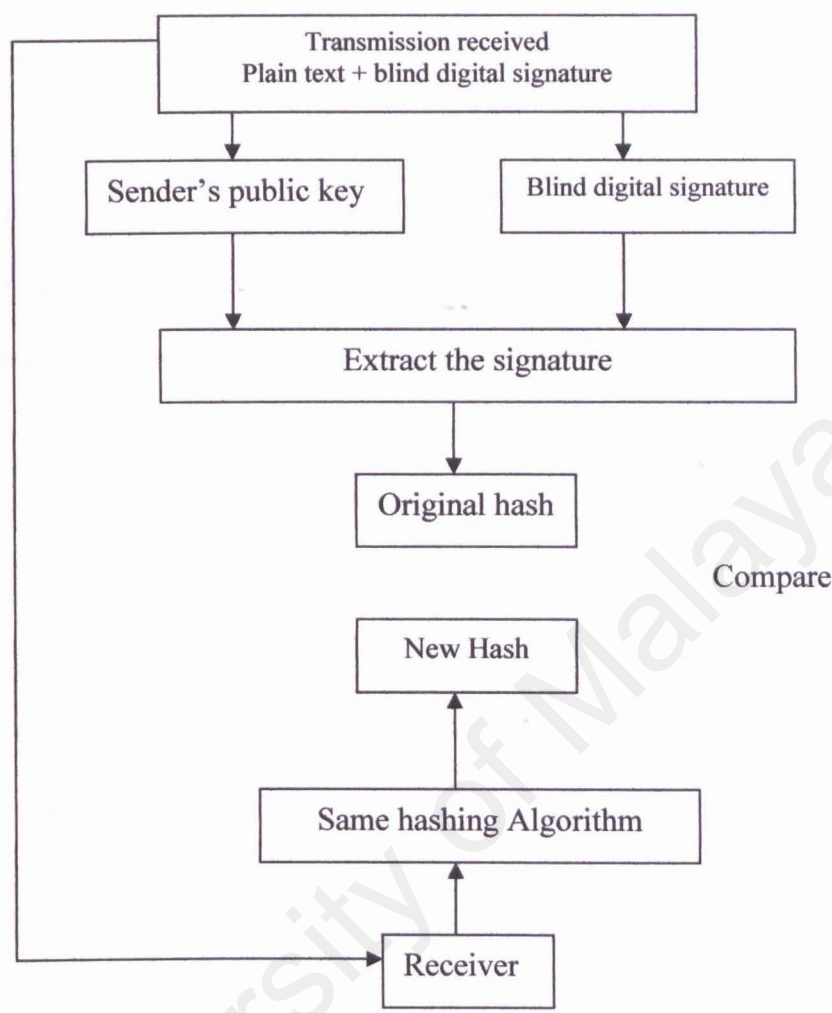


Figure 2.15 Verifying process : Receiver

2.7 Blind Signature Scheme

2.7.1 Blinding the RSA Signature Scheme

An interesting variant on the basic digital signature is blind signature. As mentioned before blind signature is a method to enable spender remained anonymous in Electronic Transaction. Such signature requires that a signer be able to sign a document without knowing it contents but when the signer is able to see the document, he should not be

able to determine when and for whom he signed it even though he can verify that the signature is indeed valid.

Let $n = pq$ where p and q are two large primes and let e be chosen such that $(e, \Phi(n)) = 1$ and $\Phi(n) = (p-1)(q-1)$. Moreover let d be such that $de = 1 \pmod{\Phi(n)}$. Assume that the signer's public key is (n, e) and the private key is (p, q, d) and finally let H be the collision resistant hash function.

Supposed that Bob requires Alice to sign a document but wants it to be the case that Alice does not know the contents of this document.

2.7.1.1 Blind signature protocol

Round 1:

- Bob wants a message M is blindly signed by Alice. Therefore, before the message is sent to Alice, Bob first blind the message by multiple the messages with random number, r :

$$M' = H(M) \cdot r^e \pmod{n}$$

Where n and e is taken from Alice's public key.

- The message then is sent to Alice.

Round 2:

- Alice then takes the message and blindly signed it:

$$C = H(M')^d \pmod{n}$$

- Observed that :

$$H(M')^d \pmod{n} = (H(M) \cdot r^e)^d \pmod{n} = H(M)^d \cdot r^{ed} \pmod{n}$$

Where $ed = 1$.

- Alice sent bank C to Bob

Round 3:

- Bob takes the signature C , given by Alice on the blinded message M' and extract an appropriate signature for M :

$$M = H(M') / r \pmod{n} = (H(M) r^e) / r \pmod{n} = H(M) r^{e-1} \pmod{n}$$

- The pair (M, M') now represent a valid message / signature pair under Alice's public key.

From the protocol the most important thing here is Alice had signed the message without knowing the content of the message. This is because the blinding factor r^e is multiplied to the message and as a result the final message just look like a random message to Alice. Then, after Bob unblind the message by dividing it with the blinding factor the message is now unrecognizable to Alice. In fact, Alice can verify the signature is indeed hers but she can severely limit in accurately determining when and for whom she signed the message.

2.7.2 Blind Schnorr Digital Signature Scheme

Another blind signature scheme is based on the scheme of Schnorr. This scheme is based on the intractability of the discrete logarithm problem, and is secure in the random oracle model. Now, firstly the author will explain the original Schnorr scheme and then show how to blind it.

2.7.2.1 The original of Schnorr Signature Scheme

Let G be a subgroup of Z^*_n of order q , for some value n and some prime q . Then, choose $g \in G$ that makes computing discrete logarithm in G difficult. Next, let $z \neq 0$ be the secret key of the signer, and $y = g^z$ be the public key. Finally, let H be a collision resistant hash function whose domain is $\{0,1\}^*$ and whose range Z_q .

For a message $m \in \{0, 1\}^*$ a pair $\{c, s\}$ is said to be a valid Schnorr signature on m if it satisfies the following verification equation:

$$C = H(m, g^s, y^c)$$

Where (m, g^s, y^c) refers to concatenation of m and $g^s y^c$

A valid Schnorr signature (c, s) on a message m can be generated by signer (who knows x) as follows:

- Choose $r \in \mathbb{Z}_q$
- Let $c = H(m, g^r)$
- Then choosing $s = r - cx \pmod{q}$ creates a valid schnorr signature

This work because :

$$g^s y^c = g^{(r-cx)} (g^x)^c = g^r \pmod{n}$$

Hence, $H(m, g^s y^c) = H(m, g^r) = c$. it turns out that schnorr signature scheme can be made blind.

2.7.2.2 Blinding the original Schnorr Signature Protocol

Signer's secret key is x , and it's public key is $y = g^x \pmod{n}$

The recipient wants to have message m blindly signed

Signer Round 1:

- Pick $r' \in \mathbb{Z}_q$
- Set $t' = g^{r'} \pmod{n}$ and send t' to the recipient

Recipient Round 2:

- Pick $\gamma, \delta \in \mathbb{Z}_q$
- Set $t = t' g^\gamma y^\delta \pmod{n}$
- Set $c = H(m, t)$

- Set $c' = c - \delta \pmod{q}$ and send it to the signer

Signer Round 3:

- Set $s' = r' - c'x \pmod{q}$ and send it to the recipient.

Recipient Round 4:

- Set $s = s' + \delta \pmod{q}$

The signature is now (c, s) . It is not hard to see why this signature is blind. The signer never get to see any information about either c or s because these values are blinded by the random blinding factors γ and δ respectively. Furthermore, the signature is valid:

$$g^s y^c = g^{(s' + \gamma)} y^{(c' + \delta)} = g^{(r' - c'x + \gamma + c'x)} y^\delta = t' g^\gamma y^\delta = t \pmod{n}$$

which means $c = H(m, t) = H(m, g^s, y^c)$

2.8 Programming Languages

Computer can only execute instruction that written in machine language. Therefore computer need a standardized communication to translate a program written by humans into a machine language which is called programming languages. Programming languages allow user or programmer to specify their requirement towards computer by writing those specifications with human language that will convert into specific machine code. Here, the author will emphasize three types of programming languages which are C++, Visual Basic and Java.

2.8.1 Microsoft Visual C++

C++ is an object-oriented programming (OOP) language that is viewed by many as the best language for creating large-scale applications. C++ is a superset of the C language. [http://searchdatabase.techtarget.com/sDefinition/0,,sid13_gci211850,00.html, 19 September 2004]. Other than Microsoft CryptoAPI, .NET Framework Cryptography

Model, there are also classes that used to implement the functions of cryptography in Visual C ++. The class is called System.Security.Cryptography. it provides normal cryptography services such as data encoding and decoding, hashing, random number generation and also message authentication.

2.8.2 Java

Java was developed by Sun Microsystems in 1995. Java is a programming language expressly designed for use in the distributed environment of the Internet. It was designed to have the "look and feel" of the C++ language, but it is simpler to use than C++ and enforces an object-oriented programming model. Java can be used to create complete applications that may run on a single computer or be distributed among servers and clients in a network. It can also be used to build a small application module or applet for use as part of a Web page. Applets make it possible for a Web page user to interact with the page. [http://searchwebservices.techtarget.com/sDefinition/0,,sid26_gci212415,00.html].

Not only that, java also consists of several components such as Java Cryptography Architecture (JCA) and Java Cryptography Extension (JCE). Java Cryptography Architecture (JCA) is designed according to implementation independence and interoperability, and algorithm independence and extensibility concepts. It means that we can use anything cryptographic services such as digital signature and message digest without worrying about the implementation details or even the algorithms. Algorithm independent is achieved by defining types of cryptographic engines (services), and defining classes that provide functionality of these cryptographic engines.

Implementation interoperability means that various implementations can work with each other, use each other's keys, or verify each other's signatures. Besides, algorithm extensibility means that new algorithms that fit in one of the supported engine classes can be added easily.

The Java Cryptography Extension (JCE) is a set of packages that provides a framework and implementations for encryption, key generation and key agreement, and Message Authentication Code (MAC) algorithms. Support for encryption includes symmetric, asymmetric, block, and stream ciphers. The software also supports secure streams and sealed objects.

2.8.3 Visual Basic

Visual Basic (VB) is a programming environment from Microsoft in which a programmer uses a graphical user interface to choose and modify preselected sections of code written in the BASIC programming language.

[http://searchvb.techtarget.com/sDefinition/0,,sid8_gci213309,00.html]

Therefore, Visual Basic allow programmer to create graphical user interface (GUIs) just by clicking with the mouse instead of writing the code. All the basic code is provided or is built in to the project.

2.9 System review

In this sub topic, the author will review the existing online voting system which using the blind signature scheme and the existing digital signature system.

2.9.1 Online Voting system (OVS)

Campus online voting system was successfully implemented at Cal State San Marcos for the election.

Features:

- Casting the Confidential Vote

The voter receive an email with their system generated password, open voting times, and a customized message from the ballot administrator. The voter can click on the secure web address in the email, provide their password and cast their vote in less than 5 minutes. After confirming their selections, the voter will get a randomly generated voter number that only they know, thus ensuring confidentiality. The system does not store any voting selections with the voter identity. Figure shows the ballot of the voting system.

Referendum & Ballot
Online Voting

Cal State San Marcos
Business Administration Major
Official Ballot - Accounting Society Elections

To vote for a candidate, click the checkbox next to the candidate's name.
If the person you wish to vote for is not listed, enter their name in the Write-In Candidate box.

[Logout](#)

President (Vote only for one candidate.)

| | |
|---|----------------------|
| <input type="checkbox"/> A. Lincoln | Info |
| <input type="checkbox"/> G. Washington | Info |
| <input type="checkbox"/> T. Jefferson | Info |
| <input type="text"/> Name of "Write-In" Candidate | |

Vice President (Vote only for one candidate.)

| | |
|---|----------------------|
| <input type="checkbox"/> A. Williams | Info |
| <input type="checkbox"/> T. Jackson | Info |
| <input type="checkbox"/> K. Robinson | Info |
| <input type="text"/> Name of "Write-In" Candidate | |

Figure 2.16 the ballot of the campus online voting system.

- Administering the election

- 1) Setting up the groups and users.

Lists of email address can easily be imported to the online voting system via a web interface. Administrator can create their own groups that will restrict or enable portions of the ballot for voters. Administrators can view summary reports or detailed reports using their specified groups. In addition, administrator can get a snap shot of voter turnout at anytime.

- 2) Creating the ballots and notifying the voters

To create ballots and notify users, the administrator follows steps:

- i) create election with open voting times
- ii) add offices, candidates and referendums
- iii) assign eligible voting groups to the offices and referendum items
- iv) confirm preview ballot and/or referendum
- v) use the system email notification feature to announce the election, password and voting instructions.

Referendum & Ballot Online Voting



Figure 2.17 Administrative side of campus online voting system

2.9.2 RemoteVote⁹

RemoteVote is a reliable e-voting platform to power sophisticated, secure electronic elections worldwide for public sector elections. Voters can cast ballots via multiple voting channels, the Internet, touchtone telephone, cellular phone, digital TV and kiosks, from anywhere in the world, within a secure environment.

Features:

- Access anyway -RemoteVote's Web-based interface provides for "anywhere" e-voting access to election administration, management, tabulation, and reporting with advanced toolset and formatting features. Full-scale elections are created in

⁹ <http://www.votehere.net/remotevote.html>

real time using credential-based administrative access, where multiple elections can be managed simultaneously. Customization tools allow for full control of ballot creation, style and layout.

- Customized Ballot - Using configurable settings in the RemoteVote toolset, ballots can be customized to meet the needs of the individual election. Colors, fonts, languages, and layout can all be customized and logos and images can be added as needed. Write-in functionality for candidates and issues is also available. With real-time ballot preview changes can be determined quickly.
- Convenience - Voters are able to cast ballots to RemoteVote via multiple voting channels: the Internet, touch-tone telephone, cellular phone, digital TV and kiosks, whichever is the most convenient for them.
- Easy-To-Use - To simplify the voting process, RemoteVote does not require voters to install complicated applications or plug-ins. When accessing RemoteVote via the Internet, voters access the election through a secure Internet connection and their completed ballots are encrypted and stored within the system.
- Security - The security of your election is important; that's why VoteHere uses patented security to protect elections and their results. Through a combination of industry-standard security practices and methodology, encryption, active election monitoring and best-of-breed IT practices, elections are safe, secure and private.

2.9.3 SafeGuard Sign & Crypt

Safeguard Sign & Crypt allow the user to digitally sign and encrypt the document. It is integrated into Microsoft Windows Explorer, Microsoft Word and Microsoft Excel. Besides, it is also a software security tool for exchanging and storing sensitive information that helps to achieve confidentiality, authenticity, integrity and non-repudiation of sensitive files.

Features:

- Signature is generated based on the public key technology
- Allow multiple signatures
- Multiple key pairs
- Support for time stamping
- Strong encryption using AES, triple-DES, IDEA
- Used RSA algorithm (up to 2048 bits) for generating the digital signature
- Used RSA 512-2048 bit, AES 128 bits, IDEA 128 bits, triple-DES 112-168 bits, DES 56 bits, RC2 40-128 bits, square 128 bit, safer 64 bit for encryption
- Used SHA-1, RIPEMD-160, MD5 for generating the message digest.

2.9.4 FileAssurity¹⁰

FileAssurity allow user to encrypt, digitally sign and decrypting any types of document (word processing document, spreadsheet, etc) to enhance the security of the document. Files can be stored securely on any media or shared securely with others. This software ensures that only authorized people can view the document that has been encrypted. Documents can be digitally signed to prevent them being altered.

Features:

- FileAssurity can ensure files to be completely removed in one step where the files are unable to be recovered again.
- FileAssurity can also automatically compress each file or archive to ensure minimum disk space is used
- It does not require the owner to buy certificates and keys from a Certificate Authority. Its built in key manager will generate self-signed keys and certificates which can be distribute to others.
- It is also easy to use where encrypt, sign, decrypt and verify process can be done just by clicking.
- Used AES algorithm with a 256 bit key for encryption and RSA algorithm with a 2048 bit key.
-

¹⁰ <http://www.artisoft.com/fileassurity.htm>

2.9.5 Code Signing for Digital IDs¹¹

VeriSign Code Signing Digital IDs enable software developers to digitally sign software and macros for secure delivery over the Internet. Users who download digitally signed Active X controls, Java applets, dynamic link libraries, .cab files, .jar files, or HTML content from software developer site can be confident that code really comes from the developer and has not been altered or corrupted since it was created and signed. After signing the code, if it is tampered with in any way, the digital signature will break and alert the users that the code has been altered and is not trustworthy. The VeriSign code signing Digital IDs is based on the public key cryptography system.

Features:

- VeriSign Code Signing Digital ID protects the software and with this protection the user or customer will be confident that the integrity of the code they download from site is intact - that it has not been tampered with or altered in transit.
- Digital IDs allow customers to identify the author of digitally signed code and contact them should an issue or query arise.
- Most browsers will not accept action commands from downloaded code unless the code is signed by a certificate from a trusted Certificate Authority, such as VeriSign.
- Code signing certificates are easy to use in conjunction with the vendor software tools that developers use to create products, macro and objects.

¹¹ <http://www.verisign.com/products-services/security-services/code-signing/digital-ids-code-signing/index.html>

2.9.6 E-Lock Prosigner¹²

E-Lock Prosigner is an off-the-shelf desktop digital signature software that integrates into MS Word, Excel and Adobe Acrobat. It allows to sign, encrypt, decrypt and validate the files of any other format externally. It uses digital certificates (X.509) to sign data and supports Microsoft, Netscape, Entrust frameworks. Besides, prosigner also profiles to automate common operation and policies to control the use of digital signatures.

Features:

- Sign documents directly from software applications

ProSigner integrates right into MS Word, Excel and Adobe Acrobat and allows you to sign documents right from within these applications. In case of Word and Excel, ProSigner provides intuitive icons and menu options, enabling you to sign/encrypt and perform other security operations with ease.

- Wizards for Signing / Encryption

ProSigner provides various Digital Signature features in a Wizard format. This guides the users through the complex world of digital signatures without compromising security without any specialized training.

¹² <http://www.elock.com/products/prosigner/>

- Signing / Encryption User Profiles

Users can create individual profiles for signing or encryption operations. Whenever they need to perform routine operations, they can access these profiles, which store all their settings and preferences. This helps them convert repetitive tasks into "one-click" processes.

- Right click security operations

ProSigner integrates seamlessly into the Windows environment. This allows users to "right-click" and performs signing and encryption operations right from their desktop or windows explorer.

- Support for Multiple signatures

Many business documents need to be signed by more than one party. To make it legally viable, all parties need to sign the exact contents. With ProSigner, multiple people can sign and approve the same document. An audit trail is also maintained which helps track approvals to the document. All signatures on the document can be independently verified. Any change to the content of the document will invalidate the previous signature.

- Batch Signing

ProSigner's Batch signing capability makes bulk signing quick and effortless and minimizes the time involved in routine signing tasks. This feature is particularly useful where several business documents require an authorized signature. The

traditional paper based method would require that he put his signature separately on each document even if they were routine in nature. ProSigner speeds up this process and allows for several such files to be selected and digitally signed in one go.

- Restrict access to confidential documents

To protect confidential documents, ProSigner uses encryption methods that allow only authorized people to decrypt and view the content. Access to contents of high-value business documents such as financial reports or Non Disclosure Agreements need to be restricted to authorized personnel only. With ProSigner documents can be encrypted for specific people, allowing only these people to view them.

- Time bound signatures

Allows for timestamping of documents as they are signed—a key feature for contract enforcement and auditing.

- PKI Independence

Works with industry-accepted public key infrastructures and seamlessly supports any X.509 digital certificate, including those issued by VeriSign, Digital Signature Trust, Entrust, RSA Security, and others.

- Security framework independence

Supports the MS-Crypto API, Netscape Security framework, and Entrust PKI.

- Algorithm

Used RSA algorithm with a 2048 bit key for encryption and generating the digital signature.

2.9.7 Summarization of the existing system

As stated before, for the blind signature system, the author had review the existing of online voting system that used the blind signature scheme. Besides, the author also reviews the existing digital signature system that operates similar to blind signature system. Below is the summarization of the algorithm that the system used.

Table 2.0 Summarization of the using algorithm on the existing system

| System | FileAssurity | SafeGuard Sign & Crypt | E-Lock Prosigner |
|----------------|--------------|------------------------|------------------|
| Features | | | |
| RSA | • | • | • |
| DSA | | | |
| DES | | • | |
| Blowfish | | | |
| AES | • | • | |
| SHA-1 | • | • | • |
| RIPEMP | | • | |
| MD5 | • | • | |
| Encryption | • | | • |
| Authentication | • | | • |
| Time Stamp | • | | • |

| | | | |
|-------------------------|---|--|--|
| User profile management | • | | |
| Secure file deletion | • | | |
| Build in key manager | • | | |
| Certificate Authority | • | | |

University of Malaya

3.1 Waterfall Model

The waterfall model is one of the software process models that help the software developer on developing a good software product. Software process model is able to guide the developer from the scratch until the product is produced. Besides, the good choice of software process model will lead to produce the product smoothly. Waterfall model is introduced by Royce on 1970. Figure 3.1 shows the waterfall model phases and the feedback loops for maintenance while the product is being developed. There are five phases which are requirements, analysis, design, implementation and postdelivery maintenance. In waterfall model, no phase is complete until the documentation of that phase is completed. Besides, in every phase of the waterfall model is testing. Testing should proceed continuously throughout the development of the software product. Therefore, in waterfall model there is no separate testing phase to be performed. Below are the descriptions for each phase:

- Requirements phase

The requirement analysis goal is to determine what are the exactly features needed that the system should operate.

- Analysis phase

In this phase, the requirements of the system is refined and analyzed to achieve a detail understanding of the requirements essential for developing the product correctly. Besides, the requirements also are defined as a system specification.

- Design phase

During this phase, the internal structure of the product is determined. The product is decomposed into modules. For each modules, algorithms are selected and data structures chosen.

- Implementation and unit testing phase

In this phase, the target software product is implemented in the chosen implementation languages which consist of a set of programs or program units. Unit testing involves verifying that each unit of program meets the specification.

- Integration and system testing.

After the program units are determined, all the units of the program are integrated and tested to ensure the target of the software product is achieved.

- Postdelivery Maintenance phase

During this phase, the system is installed. Maintenance here means to correct anything errors that occurred and the enhancement of the product that consist of changes to the specifications and the implementations of those changes.

Below are the advantages of waterfall model:

- Waterfall model allow the developer to departmentalize and managerial control. It means that a schedule can be set with deadlines for each stage of development and a product can proceed through the development process.
- Because of the testing activities is carried out constantly through out the development, the possibility to detect and correct the fault earlier is higher.
- Besides, waterfall model emphasized the planning of the product development.
- Measurable objectives which can be used for planning future projects.

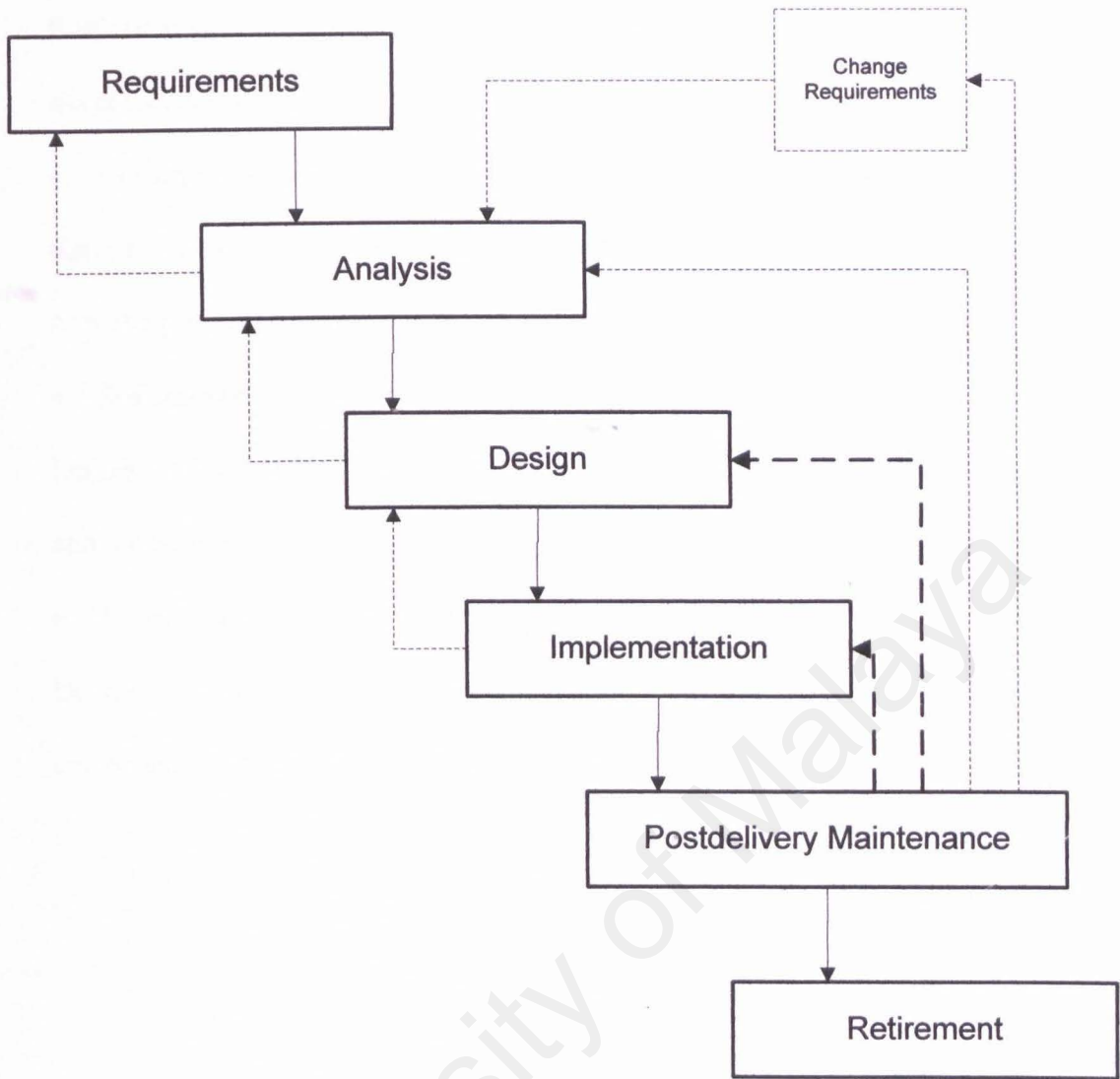


Figure 3.1 The waterfall model

3.2 Information gathering

While developing the software, the author applies a few methods on gathering as much information as a guidance to have a deep understanding on what exactly the author is trying to develop. Below are a few methods taken by the author:

- Books
- Internet

Read reference books that are related to the project to obtain the useful knowledges.

Surf the internet that provide the information needed which are up to date and ease latest technologies.

- Existing projects

Read the existing projects (the previous seniors' thesis) that give the author idea on how the project is carried out.

- Software products manual

Do research on the existing software products that are related to specific software and technologies.

- Journals and proceeding papers

Do research on journal and proceeding papers that are related to this project which can be useful guide to this project.

Chapter 4 System Analysis

The aim of this chapter is to analyze and refine requirements to achieve a detailed understanding of the requirements essential for developing a software product correctly and maintaining it easily. The requirements is categorized into two main categorizes which are system requirements and run time requirements.

4.1 System Requirements

Requirements fall into two categories which are functional and non functional requirements. Functional requirement specifies an action that the target product must be able to perform. A non functional requirement specifies properties of the target product itself such as platforms constraints, response times and reliability.

4.1.2 Functional Requirements

4.1.2.1 The signing process

- This system allowed user to browse and retrieve the data or file from user's data store.
- If the data or file length is exceed from the length that is allowed (2^{64} bits) the system will alert the user.
- Then, this system will display the selected data or file to the user before the signing and the blinding process proceeds.
- Next, the system will prompt out the private key, the public key and the random key as an input for the signing and the blinding process.
- The selected data or file is then proceed to the blinding process first before the signing process begin. The blinding process will generate the blinded data or file which is the input for the signing process.

- After the blinding process, the system will sign the blinded data or file together with the timestamp (time when the data or file is signed). Timestamp must be protected by the signature.
- During the signing process, this system is able to show the progress of the signing process and the system also is able to prompt the result of the process whether is failed or successful.
- The system will save the signed of the blinded data or file into user's data store.
- The signed data or file is represented with an icon.

Blinding file

- In this system, user is allowed to browse and retrieve the data or file that need to be blindly sign from the storage directory.
- The system then will prompt the public key to the user as the input for computing the blinding calculation.
- Besides, the system also will generate the random key for blinding process.
- Then, the blinded data or file is sent to the signing process.

Read file

- In this module, user is allowed to browse and select the data or file needed from the storage directory
- This module is only displayed the format of data or file that is accepted by the system
- This module also is able to calculate the size of the selected data or file.
- Then, this module is able to send the selected data or file to the hash file module.

Hash file

- The hash function module received a block of selected data or file which length is not exceeding from 2^{64} bits from the read module and produces a fix length (160 bits) output.
- The hash result $H(x)$ must be relatively easy to compute for any given data (x) .
- For any given h , it is computationally infeasible to find x such that $H(x) = h$.
- For any given block x , it is computationally infeasible to find data $y \neq x$ with $H(y) = H(x)$.
- The hash file module is able to send the hash result $H(x)$ to encrypt file module

Encrypt Hash

- The encrypt hash module received the hash result from the hash file module.
- In this module, the user's public key and the random key is prompted out.
- Then, the hash result of the selected data or file is encrypted. Next, the blinding process is proceed.

4.1.2.2 The Unblinding process

- This system allowed user to browse and retrieved data or file from the storage directory for specifying which data or file needed to unblind.
- Then, this system will display the selected blinded data or file to the user before the unblinding process proceed
- Before the unblinding process is proceed the system will retrieved the random key from the key data store.
- During the unblinding process, this system is able to show the progress of the unblinding process and the system also is able to prompt the result of the process whether is failed or successful.

- The unblinded data or file is saved to the user's data store.
- The unblinded data or file is represented with an icon.

Read file

- In this module, user is allowed to browse and select the blinded data or file needed from the storage directory
- This module is only displayed the format of data or file that is accepted by the system
- Then, this module is able to send the blinded selected data or file to the unblinding process.

4.1.2.3 The verifying process

- The blind signature system allowed user to browse and retrieved the data or file from the storage directory that need to be verified.
- Before the verifying process, this system is able to prompt out the user's public key as an input for the verifying process.
- During the verifying process, this system is able to show the progress of the verifying process and the system also is able to prompt the result of the process whether is failed or successful. However, if the verification process is failed the system is able to inform the user that the content of the data or file was changed or modified.

Read file

- In this module, user is allowed to browse and select the data or file needed from the storage directory

- This module is only displayed the format of data or file that is accepted by the system
- The read file module is able to extract the signature from the file and send it as an input to the decrypt blind signature module
- The read file module is able to extract the original file and send it as an input to the hash file module

Hash file

- The hash function module received a block of selected data or file which length is not exceeding from 2^{64} bits from the read module and produces a fix length (160 bits) output.
- The hash result $H(x)$ must be relatively easy to compute for any given data (x) .
- For any given h , it is computationally infeasible to find x such that $H(x) = h$.
- For any given block x , it is computationally infeasible to find data $y \neq x$ with $H(y) = H(x)$.
- The hash file module is able to send the hash result $H(x)$ to the compare hash module.

Decrypt file

- The decrypt file module received the blind signature from the read file module.
- In this module, the user's public key is prompted out.
- Then, the blind signature of the selected data or file is decrypted to recover the original hash.
- The decrypt blind signature should be able to send the decrypted blind signature to the compare hashes module.

Compare hashes

- The compare hashes module is able to compare the hash received from hash file module with the hash received decrypt blind signature module.
- The compare hashes module is also able to inform user if the signature verification is success or failed.

4.1.2.4 Key Generator Module

- The key generator module will be able to generate public key, private key and random key.

4.1.2.5 About Module

- This module provides general information about the blind signature system, user manual system and terms definition.

4.1.2.6 User Identity verification Module

- In this module, user need to log in by enter their username and password.
- This module will validate the username and password and will inform the user if the username or password is invalid.

4.1.3 Non functional Requirements

- Usability

The blind signature system is a user friendly interface where the user is eased to use. There are different icons for different purposes, menu, toolbars and pop up window as guidance for using this system.

- Reliability

The blind signature system is able to operate with minimal errors and optimum availability.

- Response Time

This system is able to operate every function that requested with a reasonable and acceptable period of time.

- Flexibility

The blind signature system allows user to change their login username and password or private and public key to avoid forgery.

4.2 Run time Requirements

The hardware and software requirements for the blind signature system are stated below.

4.2.1 Hardware Requirements

- Pentium 533 MHz or above
- 64MB RAM
- 1.44 Floppy disk drive
- Monitor 14" (high color 16 bit)

4.2.2 Software Requirements

- Microsoft Visual c++ 6.0 Professional Edition
- Windows 9x or above

4.3 Cryptography

In this blind signature system the author had chose public key cryptography after having a deep investigation in comparing between the public key (asymmetric) and secret key (symmetric) cryptography. While comparing these two types of cryptography there is several considerations that need to give attention.

- Key length

- Popularity

RSA is the most widely used and has withstood over 15 years of vigorous examination for weaknesses. Although DSS may well turn out to be strong cryptosystem, its relatively short history will leave doubts for years to come.

- Processing speed

In the context of blind signature, the faster the algorithm to verify the signature is the better. In RSA algorithm signature verification is faster than signature generation. It is differ from DSA system where the signature generation is faster than signature verification.

- Key exchange capability

RSA have a capability on key exchange.

4.4 Hash Algorithm

After a deep research on hashing algorithm which are the Secure Hashing Algorithm (SHA-1) and the MD5 algorithm, the author decided to use SHA-1 in this blind signature system. The choice is made according to several considerations as stated below. Table 4.0 below shows the different between the SHA-1 and MD5 algorithm.

Table 4.0 The differences and similarities between SHA-1 and MD5

| Algorithms | MD5 | SHA-1 |
|----------------------|--------------------|-------------------------|
| Features | | |
| Digest length | 128 bits | 160 bits |
| Unit of Processing | 512 bits | 512 bits |
| Number of steps | 64 (4 round of 16) | 80(4 round of 20) |
| Maximum message size | | 2 ⁶⁴ -1 bits |

| | | |
|-----------------------------|----|---|
| Primitive logical functions | 4 | 4 |
| Additive Constant used | 64 | 4 |

There are a few considerations while choosing the hashing algorithm:

- Resistance to Bruce Force attacks

According to the table above, SHA-1 has 160 bits as output whereas the MD5 only has 128 bits output. Therefore, because of the different in the amount bits of output, SHA-1 which has a larger amount of bits is more secure against Bruce force attacks. This is because, the attacker has difficulty on producing 2 messages which have the same message digest on the order of 2^{160} operations compare to MD5 algorithm that only need to have 2^{128} operations. Moreover, in MD5 algorithm a collision can be found by brute force in 2^{64} calculations whereas SHA-1 a collision can be found by brute force in 2^{80} calculations.

- Secure against cryptanalysis

Secure against cryptanalysis means the harder cryptanalyst to discover the weakness of the algorithm the more secure the algorithm. Therefore, in the context of cryptanalysis, MD5 algorithm is proof to have high possibility to cryptanalytic attack compare to SHA-1 algorithm.

- Speed

According to the table 4.1 shows that the speed of SHA-1 algorithm on a 266MHz is much slower than MD5 algorithm. Nevertheless, the existing of powerful hardware recently should solve the problem on the speed of the algorithm.

Table 4..1 Performance of MD5 and SHA-1 algorithm on 850 MHz Celeron

| Algorithm | Mbps |
|-----------|------|
| MD5 | 26 |
| SHA-1 | 48 |

4.5 Programming language

In this system, Microsoft visual C++ is chose instead of Java and visual basic programming language.

- Visual C ++ can perform faster than Visual Basic during the looping operations.
- Visual C ++ allows faster slipping into assembler and move the memory than the windows API Copy Memory where this is another advantage of visual C++ or Visual Basic
- Visual Basic is complained that it creates bloated installations, is not fully object oriented and performs poorly at mathematical tasks.
- Java language do not have features like hardware-specific data types, low level pointers to arbitrary memory addresses, or programming methods like operator overloading, multiple inheritance.
- Visual C++ is a graphical based language and allows more arbitration in interface design than visual Basic.
- Visual C++ has shorter, easy to understand and manipulate commands than Java.
- Visual C++ can develop platform independent software compare to visual basic which only can develop Microsoft platform software.
- Visual C++ is more object oriented than Java thus is easy to maintain.
- Visual C++ is more familiar for the author
- Optimized compiler ensures better software performance.

Chapter 5 System Design

5.1 System Architecture

The overall architecture of blind signature system is depicted with top down approach as shown in figure 5.1.

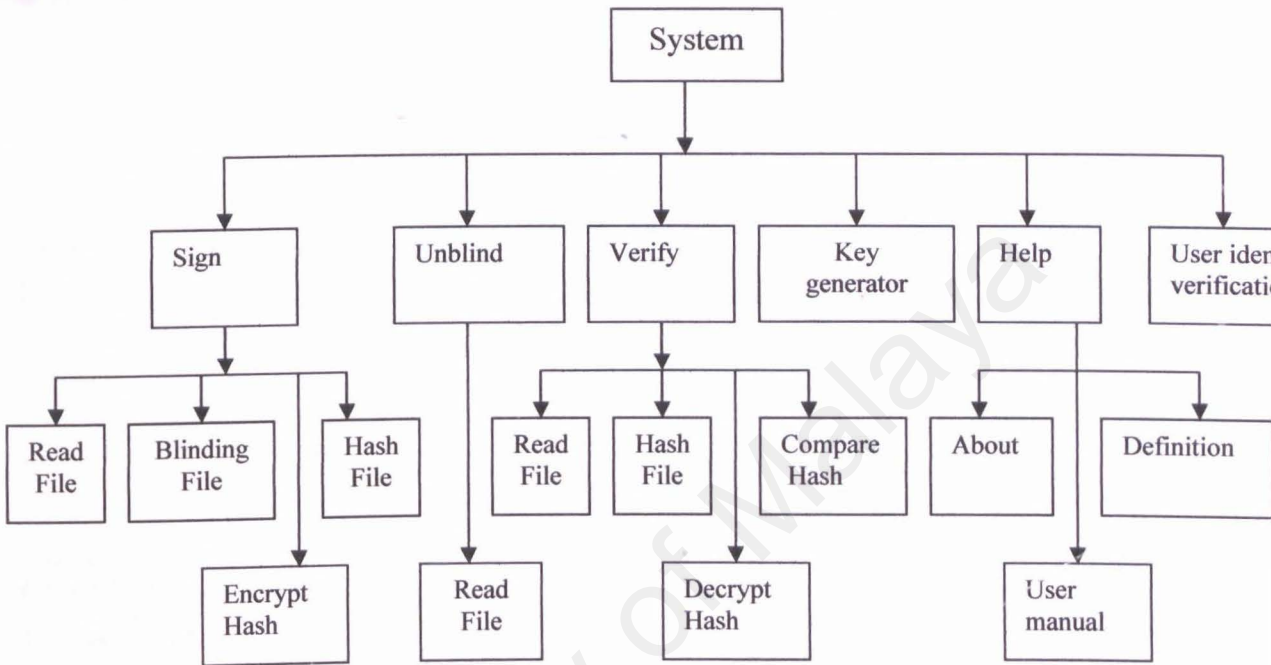


Figure 5.1 Overall System Architecture

5.2 Data Flow Diagram

Figure 5.2 show the data flow of diagram of the blind signature system. It illustrates how the data is processed and stored in the blind signature system.

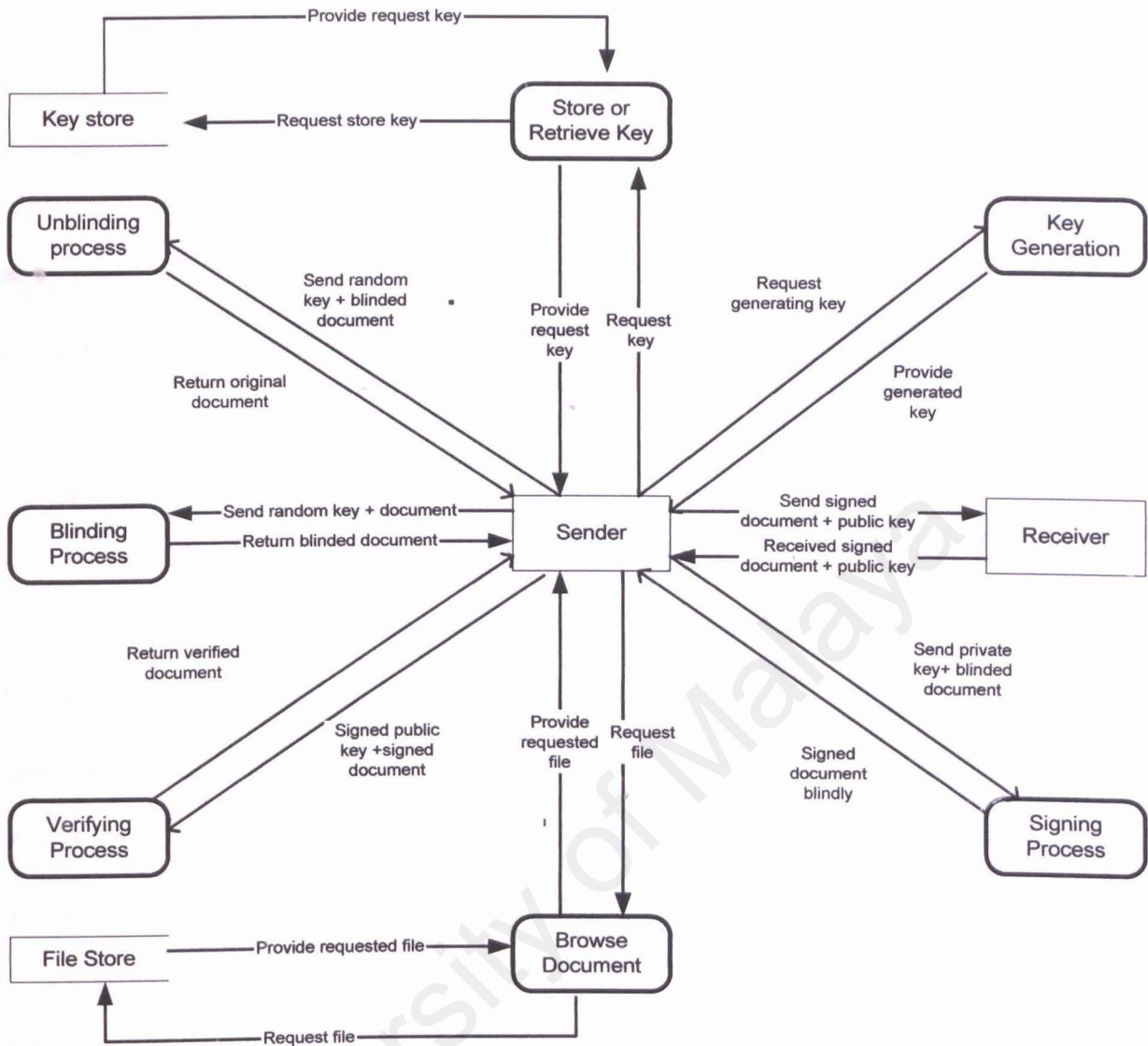


Figure 5.2 Data Flow Diagram

According to data flow diagram, the sender and the receiver are the two parties that involves in this blind signature system. The sender, who triggers the blindly signing activities, will browse the document needed and the document is retrieved from a file store. Then, the sender can choose to generate a new private key, public key and a random key or just retrieved the existing key from the key store. The sender then, firstly, will blind the document. The blinding process will then generate a blinded document after it received the random key and the document that need to be blind. After, the

blinding process is completed the sender now is proceed to the signing process to generate a signed document. The signed document is only generated when the blinded document that needs to be signed is retrieved and the private key is received. The unblinding process is completed after the blinded document that posses the valid signature is retrieved and the random key is received. The unblind document then will proceed to verifying process. In the verifying process, the receiver is the one who triggers the verifying activities. During the activities, the process will inform the sender if the signature on the document is indeed valid or not.

5.3 Interface design

Figure 5.3 shows the system main interface prototype design. According to the interface design, user firstly needs to click on key generator button to generate the keys. The pop menu that shows the keys is prompt out as shows in figure 5.4. After the key is obtained from the system, user now can choose any task on the interface that the user want the system to execute. If the user clicks on the sign button the new window of the sign process is appeared. It shows on figure 5.5. These interfaces are user friendly and have a direct manipulation which provides menus such as pop up menu, iconic and cascading menu.

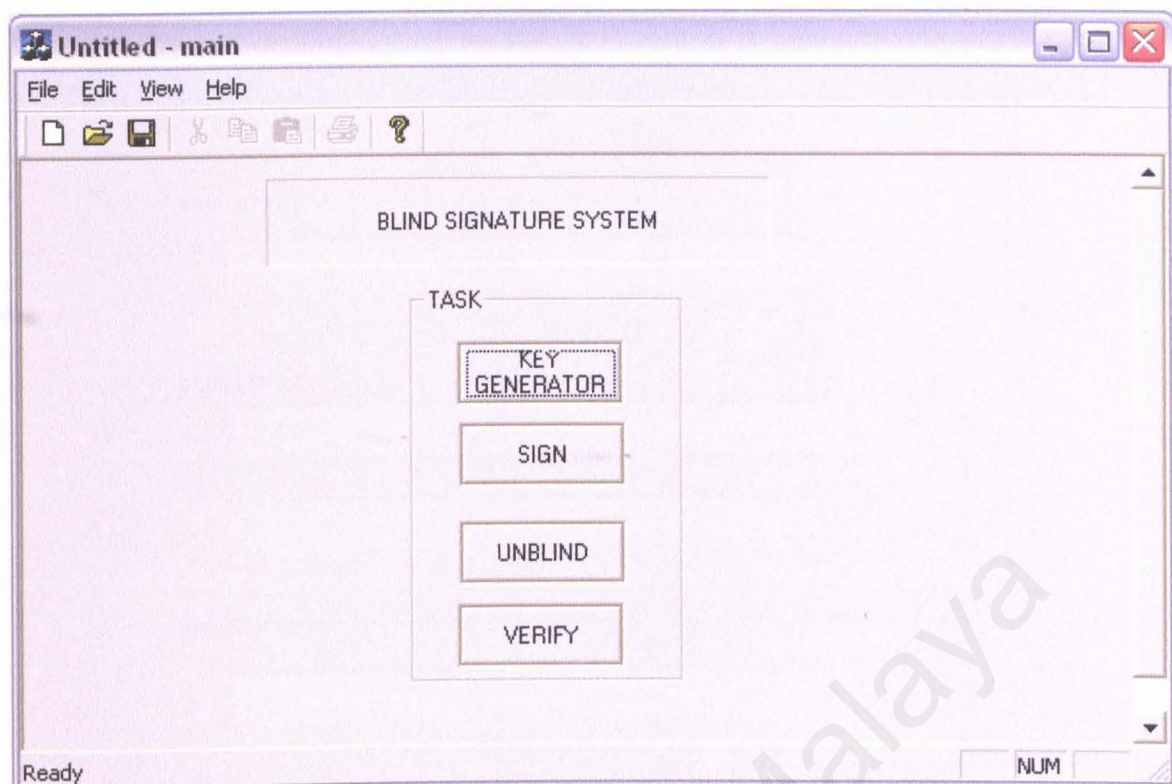


Figure 5.3 The System Main Interface Prototype Design

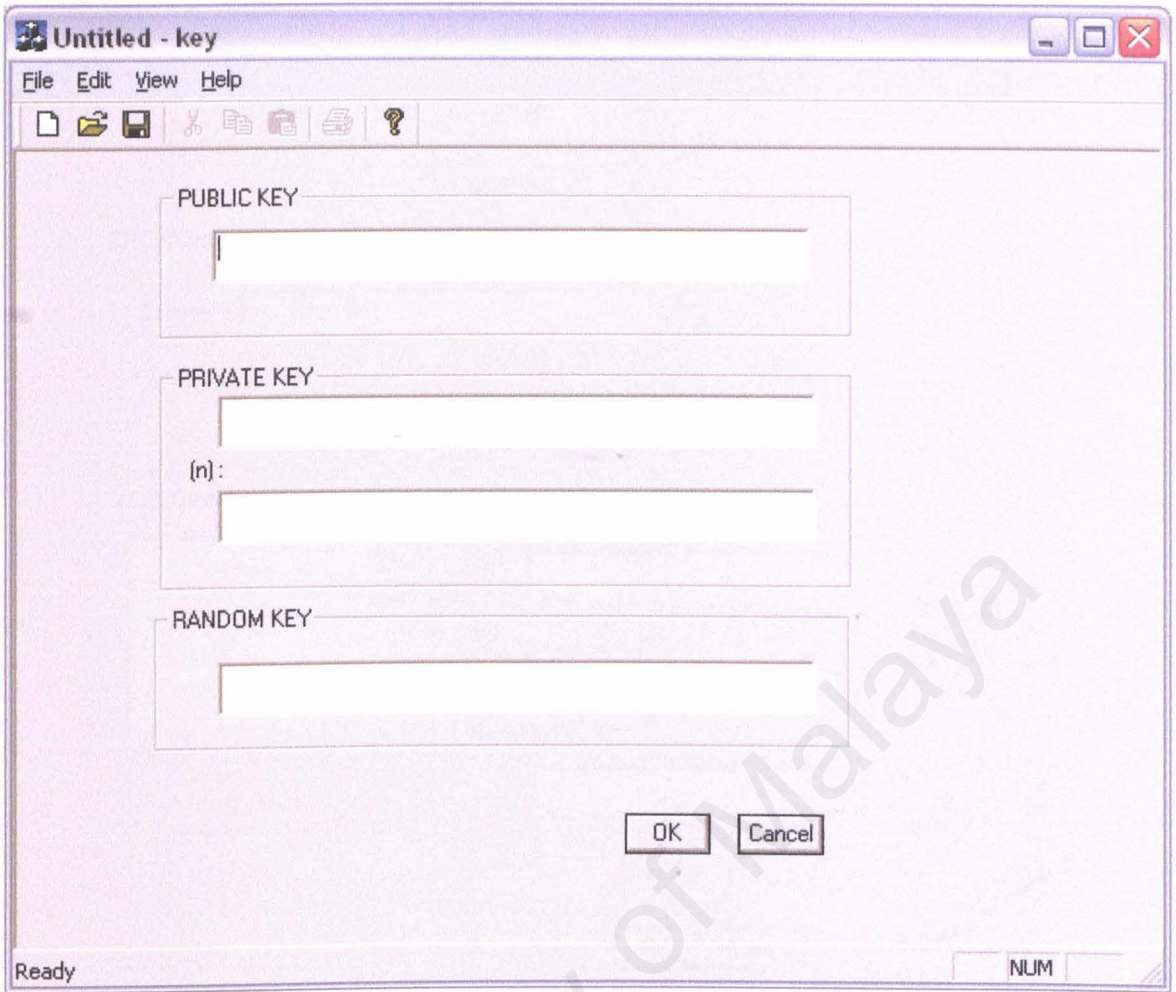


Figure 5.4 Key Pop Up Menu Prototype Design

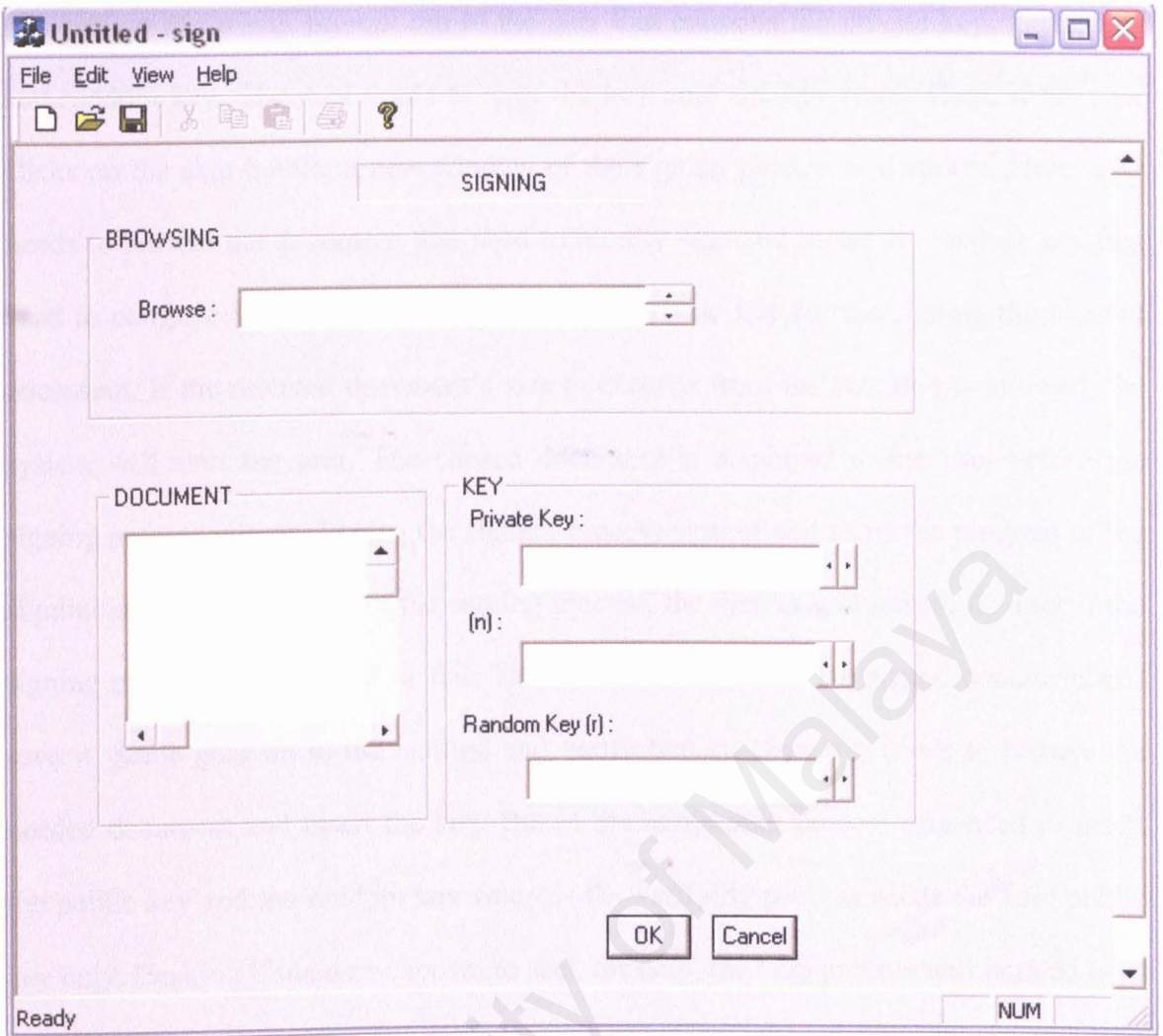


Figure 5.5 Signing Process Interface Prototypes Design

5.4 Interface flow chart

Figure 5.6 shows the interface flow chart of the blind signature system. Firstly, user need to key in their username and password in a dialog box that is prompted out to the user. If the username or password is not valid one error message will prompt out to the user and user need to insert the username and password again. After the log in session is successful, the system will display the main menu of the blind signature system. In this system, there are 6 tasks that the user can choose which are sign, verify, unblind, generate key, help and close the system. If the user clicks on the key generator button,

one pop up menu will prompt out to the user that contains the private key, public key and random key. The user needs to save the key onto the key store. Then, if the user clicks on the sign button, a new window of the signing process will appear. Here, user needs to browse the document that need to blindly sign and insert the random key that used to compute the blinded document and the private key for the signing the blinded document. If the selected document's size is exceeds from the size that is allowed, the system will alert the user. The chosen document is displayed to the user before the signing process initiate. During the signing process system will show the progress of the signing process. At the end of the signing process, the system will inform the user if the signing process is successful or fail. Then, the user can view the signed document and save it. Same goes on to the unblind and verify button. User still needs to browse the needed document and insert the key. But in the unblinding process user need to insert the public key and the random key whereas the verifying process needs the user public key only. Besides, if the user chooses to seek for help, the help process will iterated until the users' queries are solved or until the user itself abort the process. Finally, the user can also choose to close the system.

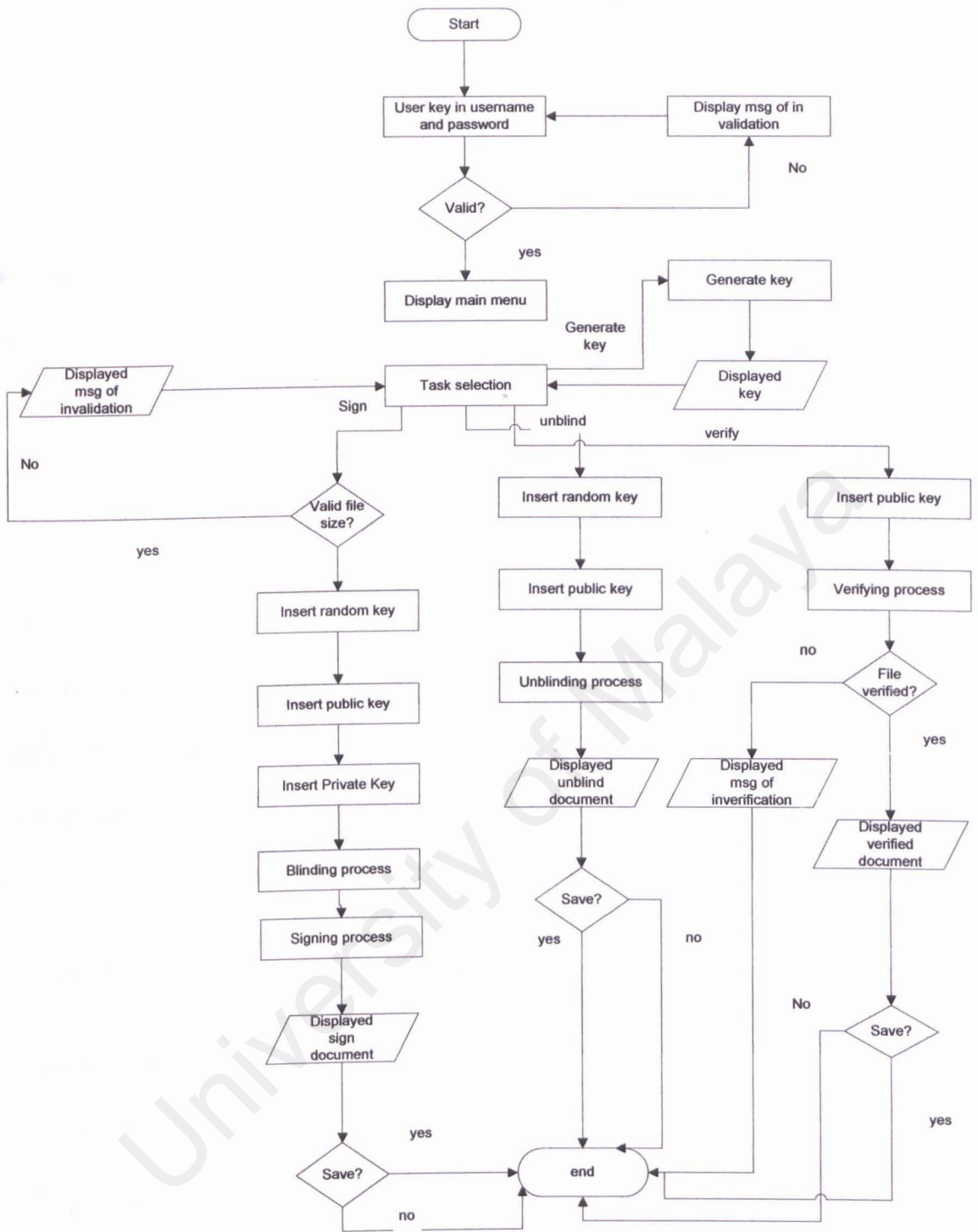


Figure 5.6 The Interface Flow Chart of the Blind Signature System

Chapter 6 System Implementation

6.1 Introduction

System implementation is the implementation of the target software product in the chosen implementation languages. In other words, system implementation is a process that converts the system requirements and designs into program codes. It is the delivery of the system into production, which means day-to-day operation.

6.2 Development Environment

Development Environment is used to determine whether the requirements of the hardware and software that were stated in chapter 4 are suitable for the software implementation. This is important because the usage of appropriate software or hardware will influence the development of the software product. In this section, the hardware and software tools used to develop the entire system are listed below.

6.2.1 Hardware Tools

- Intel (R) Pentium (M) processor 1500 Mhz
- 256 MB RAM
- 40 Gbytes Hardisk
- BJC i255 Cannon Printer

6.2.2 Software Tools

- Microsoft Windows XP Home Edition
- Microsoft Visual C++ 6.0 Professional Edition

6.3 System Development Tools

In developing the blind digital signature program, the author used Microsoft Visual C++ 6.0 as a tool to generate the coding and the interfaces of the program. One of the functions that Microsoft Visual C++ 6.0 provides is Microsoft Foundation Class (MFC) that is used to design the interfaces. Therefore, in this section, it will be the user interfaces development and coding development.

6.3.1 User Interface Development

User interface is the part of a computer program that displays on the screen for the user to see. It will describe how humans interact with what they see on the computer screen. Besides, the good interface will help the user to understand on how to use the program accurately. In this section, the real implementation of the actual user interface will be discussed.

6.3.2 User Authentication Dialog

6.3.2.1 Log In Dialog

Log in dialog is used to authenticate the user and it will pop up to the screen once the icon of the program is clicked. The user is prompted to key in the user ID and the password in order to enter the program as shown in figure 6.1.

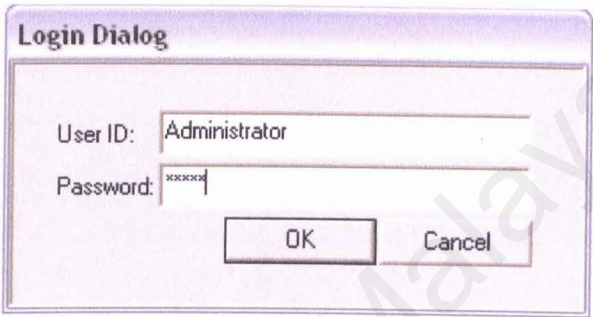


Figure 6.1 Log in dialog

6.3.2.2 Error message

If the user had key in either the wrong user ID or password the error message will pop up to the user as shown in figure 6.2.



Figure 6.2 Error Message

6.3.2.3 Blind Key Input Dialog

Before blinding the message user is required to enter the random key, public key and multiple value that are generated by the program as shown in figure 6.3.

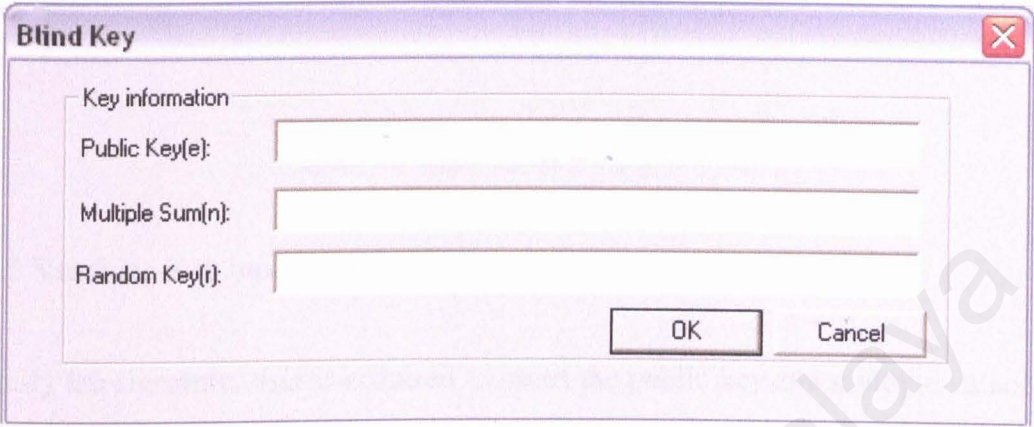
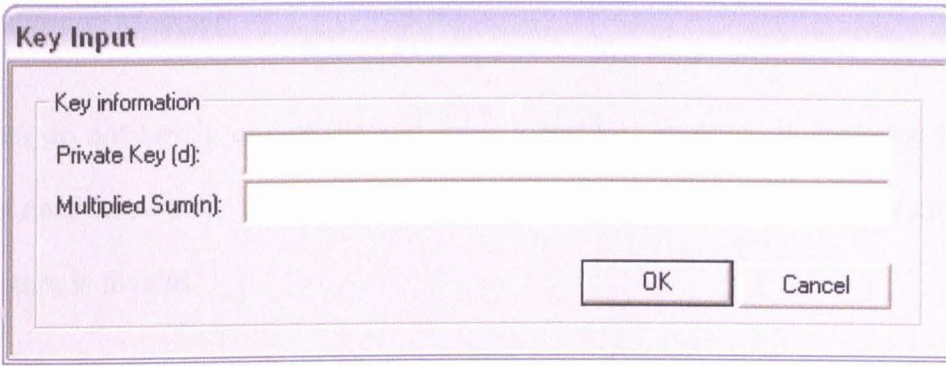


Figure 6.3 Blind Key Input Dialog

6.3.2.4 Private Key Input Dialog

After blinding the message the user can proceed to sign the blinding message. Same goes here; the user is acquired to key in the private key and the multiple value that also are generated by the program itself.

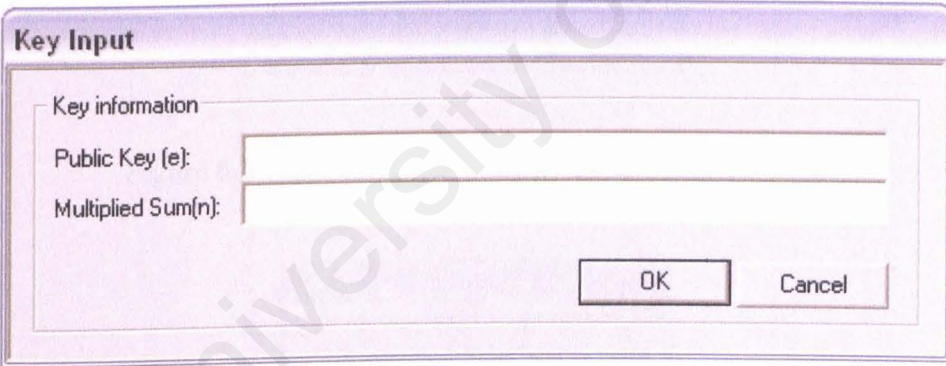


A screenshot of a Windows-style dialog box titled "Key Input". It contains a group box labeled "Key information" with two text input fields: "Private Key (d):" and "Multiplied Sum(n):". At the bottom right are "OK" and "Cancel" buttons.

Figure 6.4 Signing Key Input Dialog

6.3.2.5 Verifying Key Input Dialog

To verify the signature, user is acquired to insert the public key and multiple values. The public key, private key, random key and multiple value are the key that are generated at the same time during the key generation.



A screenshot of a Windows-style dialog box titled "Key Input". It contains a group box labeled "Key information" with two text input fields: "Public Key (e):" and "Multiplied Sum(n):". At the bottom right are "OK" and "Cancel" buttons.

figure 6.5 Verifying Key Input Dialog

6.2.3.6 Warning Message

If the user do not key in one of the key value a warning message is appeared to inform that they need to re-enter the key value. Besides, a warning message also is appeared if the signature is invalid.

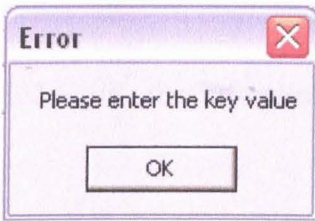


Figure 6.6 Error message Dialog box (Public key and Private key)

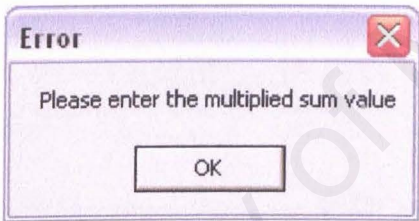


Figure 6.7 Error message Dialog Box (Multiple value)

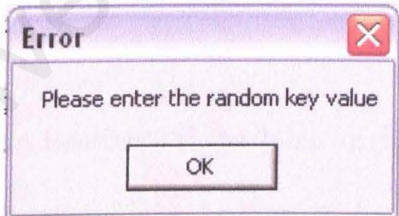


Figure 6.8 Error message Dialog Box (Random key)



Figure 6.9 Error message Invalid Signature

Despite, if the signature is indeed valid, the informing message that states the valid signature is appeared.



Figure 6.10 Informing message Valid Signature

6.3.3 Main Interfaces

Figure 6.11 below is the main interface of the blind digital signature program. It will emerge once the user id and password is authenticated. As shown in the main interfaces, there are seven tasks that hold different functions.

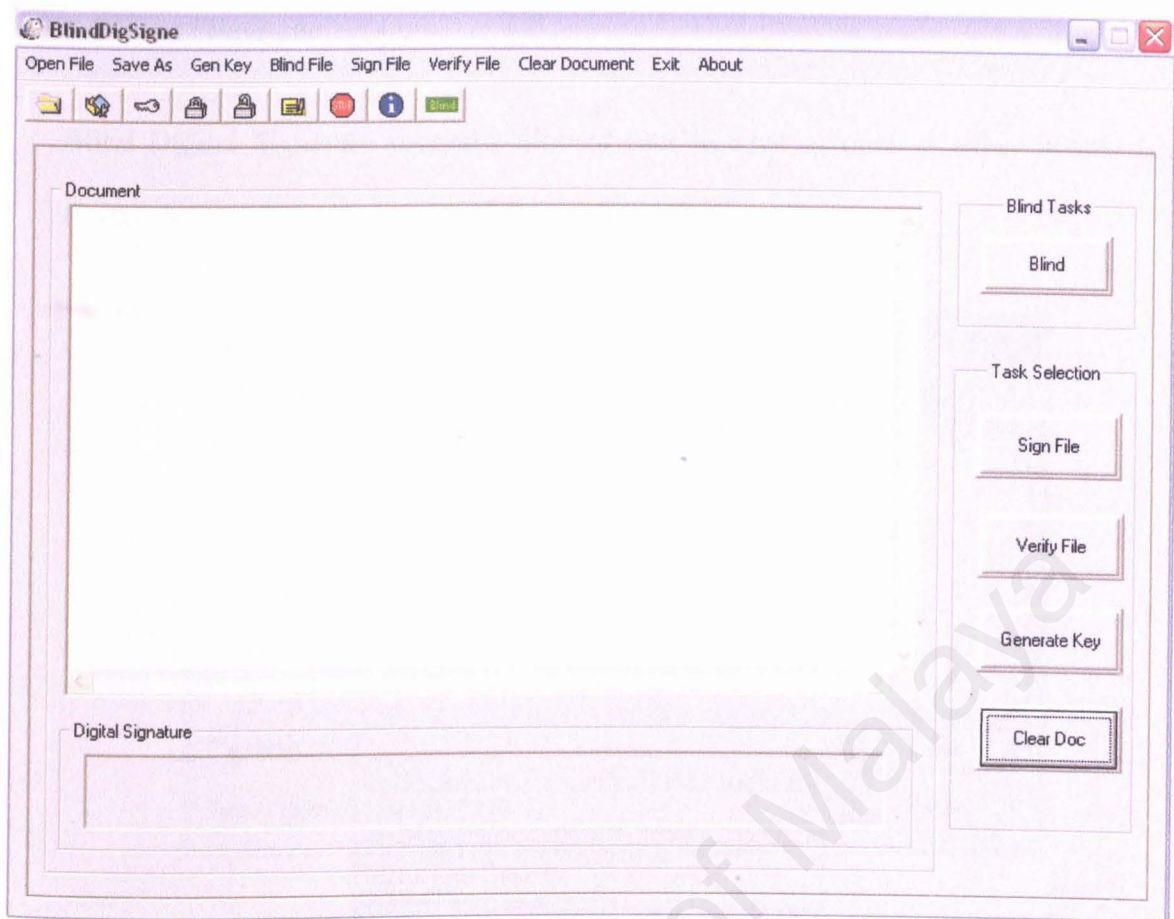


Figure 6.11 Main Interface

6.3.3.1 Open / Save file

Blind Digital Signature program allowed user to save or open whether in text file (*.txt), Signed Files (*.signe) or Blinded Files (*.blinded).

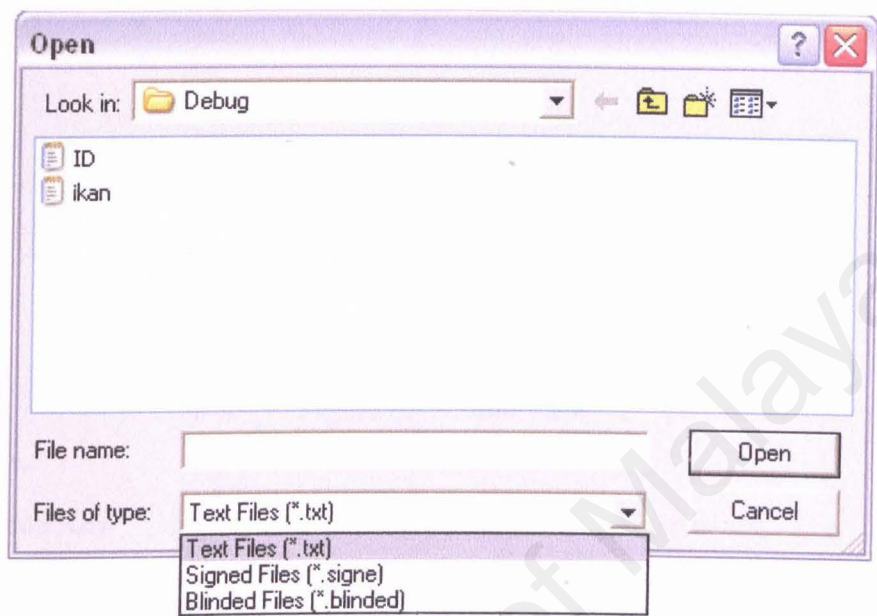


Figure 6.12 Open File Task

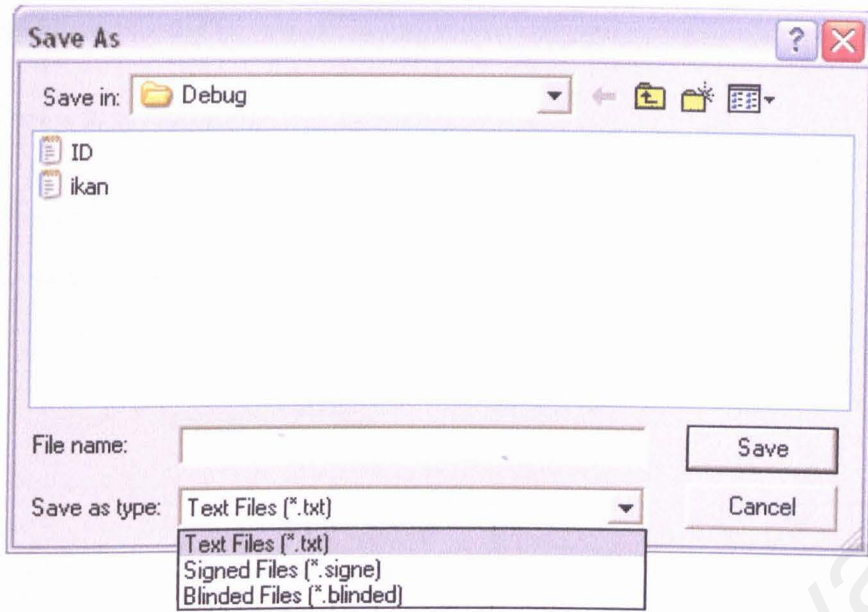


Figure 6.13 Save Task

6.3.3.2 Generate Key

All keys are generated at the same time by the key generator. User need to copy and save all the keys somewhere else because this program do not have a special built in databases that can keep them secretly.

Generated Keys

Private Key (to be kept secret)

Private Key (d): 835B220300150D06

Random Key(r): 00000053

Public Key Components (to be distributed)

Public Key (e): 0000C353

Multiplied Sum (n): 12817003002D3592

OK

Figure 6.14 Key Generator

6.3.3.3 Blinding Task

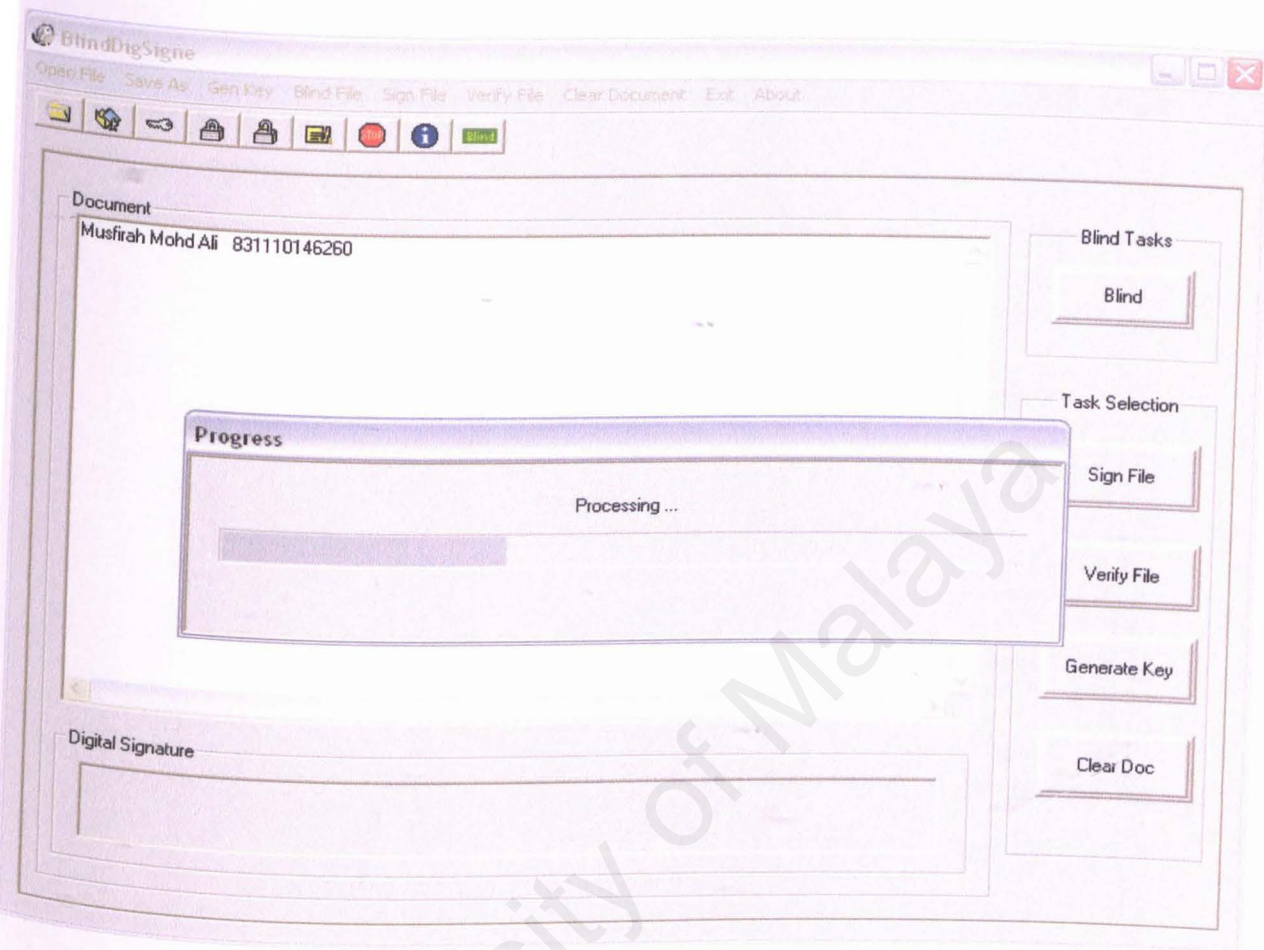


Figure 6.15 Blinding Process

6.3.3.4 Signing Task

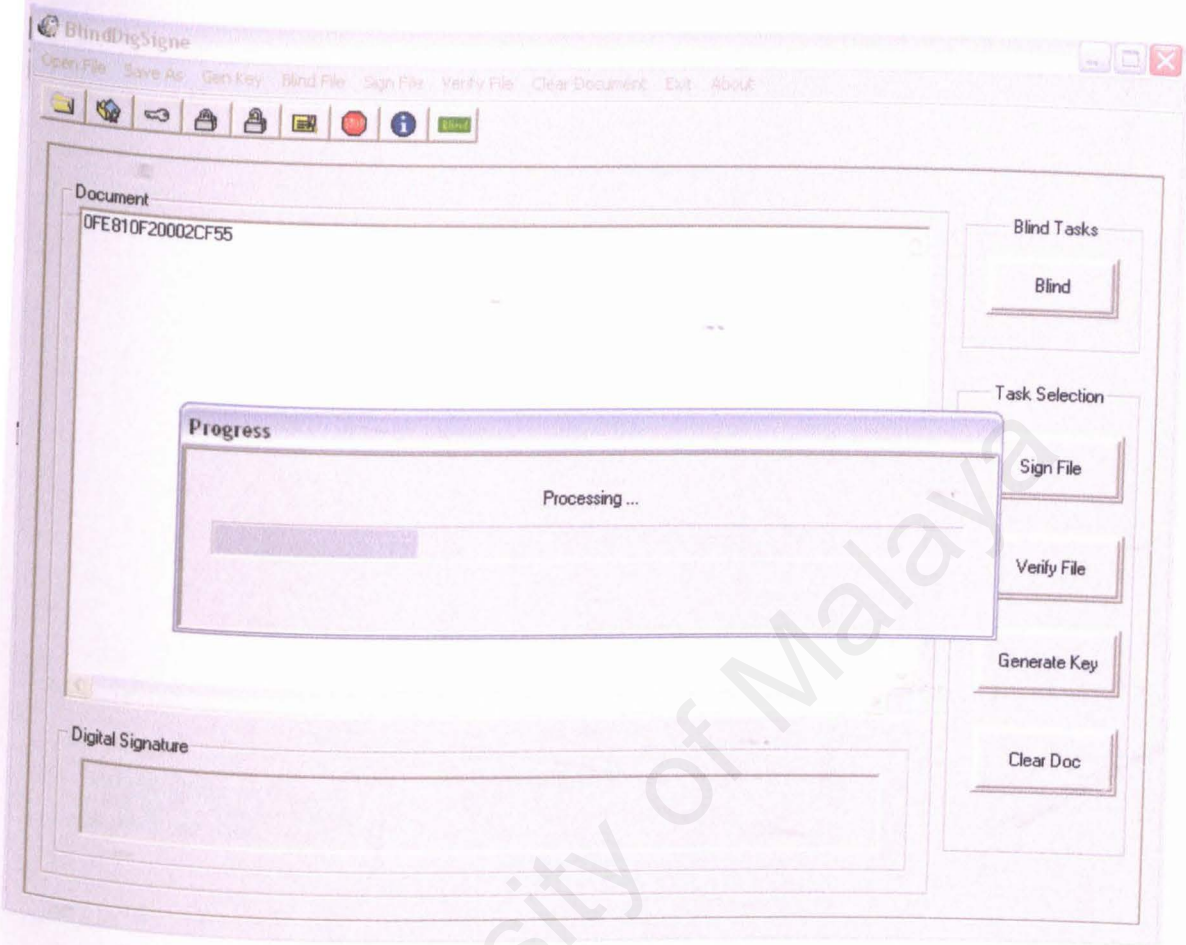


Figure 6.16 Signing Process

6.3.3.5 Verifying Task

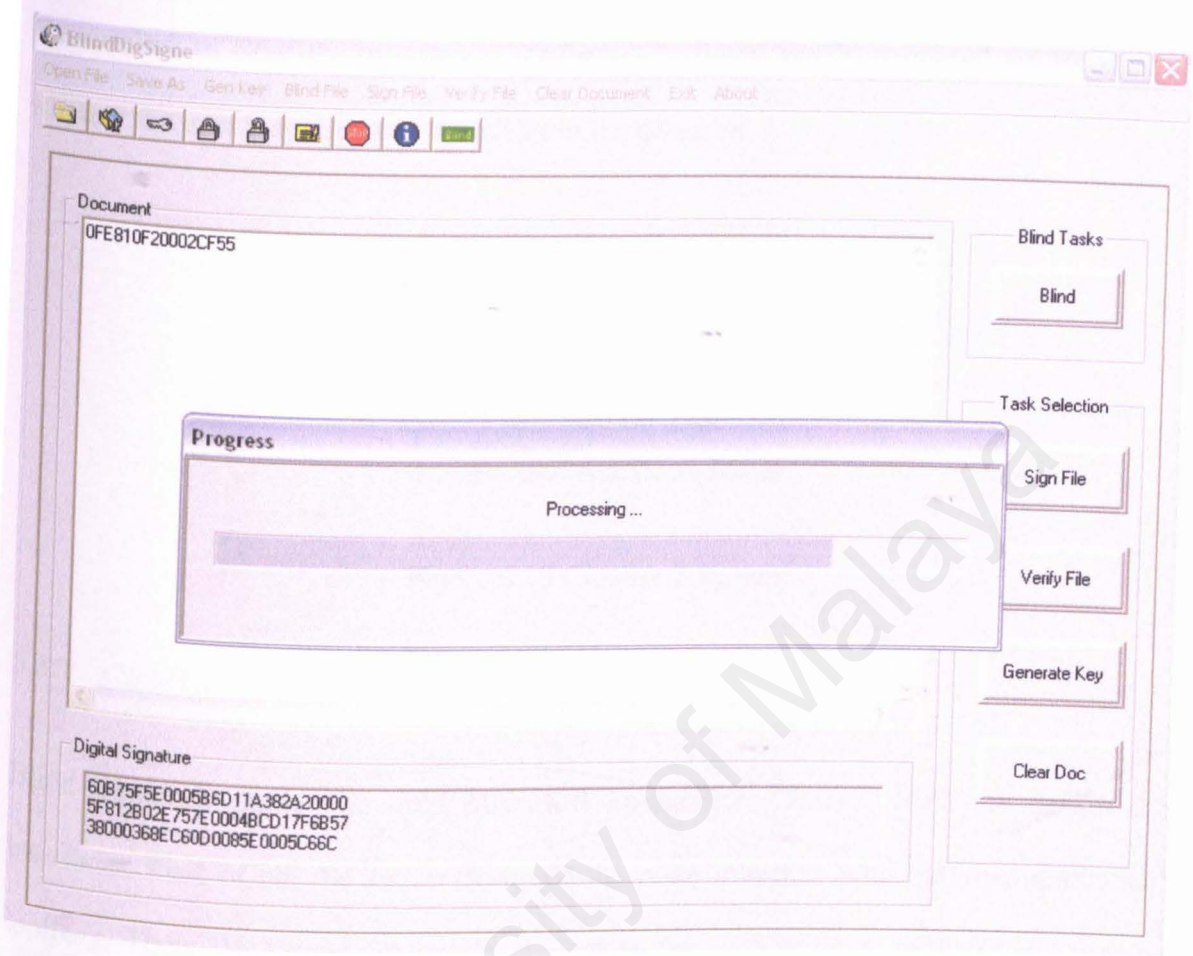


Figure 6.17 Verifying Process

6.3.3.6 Exit

If user click to close the program, a confirmation dialog box is prompted out to confirm whether the user is really want to exit from the program.

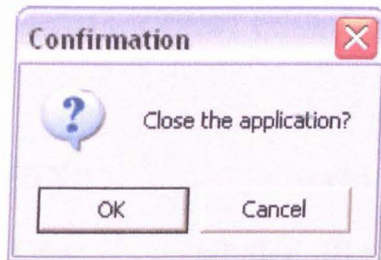


Figure 6.18 Closing Program

6.3.2 Code Development

Blind signature application used Microsoft Foundation Classes (MFC) to design the interfaces. First of all, the author designed the main interface with buttons, menus and toolbars. Then, the author determines the action for each buttons, toolbars and menus. Every action has their own function that contains lines of coding. Coding is the most important thing that needs to be done. This is because a successful coding determines the strength of the application. Here, the author will explain the coding of hash function and blinding function

i) Hashing Function

- ProcessSHA()

/* This function is to chop the plaintext into appropriate size and to collect the Message digest in order to make available to the function of blinding and decryption.*/

```
void CDigSignDlg::ProcessSHA( CONST CString& strData, CString& strHash )
```

```
{  
  
    SHA1_CTX          context;  
  
    unsigned char      aBuffer[ SHA_BUFFER ];  
  
    unsigned char      digest[20];  
  
    CString strTemp = strData;  
  
    CString strBlock = _T("");  
  
    Int nDataLen = strTemp.GetLength();  
  
    Int nIndex = 0;  
  
    CSHA1::SHA1Init( &context );  
  
    while ( nDataLen > 0 ){  
        memset( aBuffer, '\0', SHA_BUFFER );  
  
        if ( nDataLen > SHA_BUFFER ) {  
            strBlock = strTemp.Left( SHA_BUFFER );  
            strTemp = strTemp.Mid( SHA_BUFFER );  
        } else  
        { strBlock = strTemp;  
          strTemp = _T("");  
        }  
  
        memcpy( aBuffer, LPCTSTR(strBlock), strBlock.GetLength() );  
    }  
}
```

```

CSHA1::SHA1Update( & context, aBuffer, strBlock.GetLength() );

nDataLen = strTemp.GetLength(); }

CSHA1::SHA1Final( digest, & context );

strHash = _T("");

for ( nIndex = 0; nIndex < 20; nIndex++ ){

    strTemp.Format( _T("%.2X"), digest[ nIndex ] );

    strHash += strTemp; }

cout<<strHash; }

```

Explanation for variables

strData : Original data or plaintext
 strHash : Storage for message digest in digit
 aBuffer[16384]: A temporary storage for characters to be process
 digest[20] : Storage for 160-bit message digest in hexadecimal
 strTemp : Temporary storage for string characters
 nDataLen : Storage for the length of strTemp
 nIndex : Counter

Pseudo Code

1. Start
2. Initialization
 - 2.1 Copy original data (plaintext) into strTemp
 - 2.2 Initialize strBlock with empty string
 - 2.3 Store into nDataLen the number of characters of strTemp
 - 2.4 Initialize the counter with 0
3. while nDataLen>0, do
 - 3.1 Set aBuffer with a total of 16384 '0' characters
 - 3.2 If nDataLen>16384, do
 - 3.2.1 First copy the leftmost 16384 characters of strTemp into strBlock
 - 3.2.2 Then, copy the rest characters into strTemp

- 3.3 If `nDataLen < 16384`, do
 - 3.3.1 First, copy into `strBlock` the data in `strTemp`
 - 3.3.2 Then, empty the `strTemp`
- 3.4 Copy `strBlock` into `aBuffer`
- 3.5 Call function `SHA1Update(&context, aBuffer, strBlock.GetLength())`
- 3.6 Store into `nDataLen` the remaining number of characters in `strTemp`
4. Call function `SHA1Final(digest, &context)`
5. Initialize `strHash` – T(“ “)
6. If `nIndex < 20`, do
 - 6.1 Change the format of digest (hex) into correspondent string of digits and store in `strTemp`.
 - 6.2 Add `strTemp` into `strHash`
 - 6.3 `nIndex`
 - 6.4 Go to step 6
7. End

• SHAInit()

/* The function of SHAInit is to make initialization to MD buffer before each 512-bit data block is process */

```
void CSHA1::SHA1Init(SHA1_CTX* context){
    /* SHA1 initialization constants */
    context->state[0] = 0x67452301;
    context->state[1] = 0xEFCDAB89;
    context->state[2] = 0x98BADCFE;
    context->state[3] = 0x10325476;
    context->state[4] = 0xC3D2E1F0;
    context->count[0] = context->count[1] = 0; }
```

Explanation of Variables

Context is a structure that contains

`state[5]` : To store intermediate and final results of the hash function, also known as message digest(MD) buffer.

Count[2] : To store bits counted in each block data before and after bits padding

Buffer[64] : To store data to be processed by hash function

Pseudo code

1. Start
2. Initialization in hexadecimal format
 - 2.1 state[0] = 67452301
 - 2.2 state[1] = EFCDAB89
 - 2.3 state[2] = 98BADCFE
 - 2.4 state[3] = 10325476
 - 2.5 state[4] = C3D2EIF0
 - 2.6 count[0] = count[1] - 0
3. End

• SHA1Transform()

/* The function of SHA1Transform is to execute the core hash function in order to produce intermediate and final message digest*/

```
#include "SHA1.hxx"
/* #define LITTLE_ENDIAN * This should be #define'd if true. */
/* #define SHA1HANDSOFF * Copies data before messing with it. */

#define LITTLE_ENDIAN 1

#define rol(value, bits) (((value) << (bits)) | ((value) >> (32 - (bits))))

/* blk0() and blk() perform the initial expand. */
/* I got the idea of expanding during the round function from SSLeay */
#ifdef LITTLE_ENDIAN
#define blk0(i) (block->l[i] = (rol(block->l[i],24)&0xFF00FF00) |(rol(block->l[i],8)&0x00FF00FF))
#else
#define blk0(i) block->l[i]
#endif
#define blk(i) (block->l[i&15] = rol(block->l[(i+13)&15]^block->l[(i+8)&15]^block->l[(i+2)&15]^block->l[i&15],1))

/* (R0+R1), R2, R3, R4 are the different operations used in SHA1 */
#define R0(v,w,x,y,z,i) z+=((w&(x^y))^y)+blk0(i)+0x5A827999+rol(v,5);w=rol(w,30);
#define R1(v,w,x,y,z,i) z+=((w&(x^y))^y)+blk(i)+0x5A827999+rol(v,5);w=rol(w,30);
#define R2(v,w,x,y,z,i) z+=(w^x^y)+blk(i)+0x6ED9EBA1+rol(v,5);w=rol(w,30);
```

```
#define R3(v,w,x,y,z,i)
z+=(((w|x)&y)|(w&x))+blk(i)+0x8F1BBCDC+rol(v,5);w=rol(w,30);
#define R4(v,w,x,y,z,i) z+=(w^x^y)+blk(i)+0xCA62C1D6+rol(v,5);w=rol(w,30);
```

```
/* Hash a single 512-bit block. This is the core of the algorithm. */
```

```
void CSHA1::SHA1Transform(SHA1_CTX* context, unsigned char buffer[64])
{
```

```
    unsigned long a, b, c, d, e;
```

```
    typedef union {
        unsigned char c[64];
        unsigned long l[16];
    } CHAR64LONG16;
```

```
    CHAR64LONG16* block;
```

```
#ifndef SHA1HANDSOFF
```

```
    static unsigned char workspace[64];
    block = (CHAR64LONG16*)workspace;
    memcpy(block, buffer, 64);
```

```
#else
```

```
    block = (CHAR64LONG16*)buffer;
```

```
#endif
```

```
    /* Copy context->state[] to working vars */
```

```
    a = context->state[0];
```

```
    b = context->state[1];
```

```
    c = context->state[2];
```

```
    d = context->state[3];
```

```
    e = context->state[4];
```

```
    /* 4 rounds of 20 operations each. Loop unrolled. */
```

```
R0(a,b,c,d,e, 0); R0(e,a,b,c,d, 1); R0(d,e,a,b,c, 2); R0(c,d,e,a,b, 3);
R0(b,c,d,e,a, 4); R0(a,b,c,d,e, 5); R0(e,a,b,c,d, 6); R0(d,e,a,b,c, 7);
R0(c,d,e,a,b, 8); R0(b,c,d,e,a, 9); R0(a,b,c,d,e,10); R0(e,a,b,c,d,11);
R0(d,e,a,b,c,12); R0(c,d,e,a,b,13); R0(b,c,d,e,a,14); R0(a,b,c,d,e,15);
R1(e,a,b,c,d,16); R1(d,e,a,b,c,17); R1(c,d,e,a,b,18); R1(b,c,d,e,a,19);
R2(a,b,c,d,e,20); R2(e,a,b,c,d,21); R2(d,e,a,b,c,22); R2(c,d,e,a,b,23);
R2(b,c,d,e,a,24); R2(a,b,c,d,e,25); R2(e,a,b,c,d,26); R2(d,e,a,b,c,27);
R2(c,d,e,a,b,28); R2(b,c,d,e,a,29); R2(a,b,c,d,e,30); R2(e,a,b,c,d,31);
R2(d,e,a,b,c,32); R2(c,d,e,a,b,33); R2(b,c,d,e,a,34); R2(a,b,c,d,e,35);
R2(e,a,b,c,d,36); R2(d,e,a,b,c,37); R2(c,d,e,a,b,38); R2(b,c,d,e,a,39);
R3(a,b,c,d,e,40); R3(e,a,b,c,d,41); R3(d,e,a,b,c,42); R3(c,d,e,a,b,43);
R3(b,c,d,e,a,44); R3(a,b,c,d,e,45); R3(e,a,b,c,d,46); R3(d,e,a,b,c,47);
R3(c,d,e,a,b,48); R3(b,c,d,e,a,49); R3(a,b,c,d,e,50); R3(e,a,b,c,d,51);
R3(d,e,a,b,c,52); R3(c,d,e,a,b,53); R3(b,c,d,e,a,54); R3(a,b,c,d,e,55);
```



```

R3(e,a,b,c,d,56); R3(d,e,a,b,c,57); R3(c,d,e,a,b,58); R3(b,c,d,e,a,59);
R4(a,b,c,d,e,60); R4(e,a,b,c,d,61); R4(d,e,a,b,c,62); R4(c,d,e,a,b,63);
R4(b,c,d,e,a,64); R4(a,b,c,d,e,65); R4(e,a,b,c,d,66); R4(d,e,a,b,c,67);
R4(c,d,e,a,b,68); R4(b,c,d,e,a,69); R4(a,b,c,d,e,70); R4(e,a,b,c,d,71);
R4(d,e,a,b,c,72); R4(c,d,e,a,b,73); R4(b,c,d,e,a,74); R4(a,b,c,d,e,75);
R4(e,a,b,c,d,76); R4(d,e,a,b,c,77); R4(c,d,e,a,b,78); R4(b,c,d,e,a,79);

```

```

/* Add the working vars back into context.state[] */

```

```

context->state[0] += a;

```

```

context->state[1] += b;

```

```

context->state[2] += c;

```

```

context->state[3] += d;

```

```

context->state[4] += e;

```

```

/* Wipe variables */

```

```

a = b = c = d = e = 0;

```

```

}

```

Explanation Variables

a, b, c, d, e are working variables instead of using context->state[5]

Pseudo code

1. Start
2. Initialization
 - 2.1 a->state[0]
 - 2.2 b->state[1]
 - 2.3 c->state[2]
 - 2.4 d->state[3]
 - 2.5 e->state[4]
3. Hash Function
 - 3.1 For round 0 downto 15
 - 3.1.1 execute R0
 - 3.2 For round 16 downto 19
 - 3.2.1 execute R1
 - 3.3 For round 20 downto 39
 - 3.3.1 execute R2
 - 3.4 For round 40 downto 59
 - 3.4.1 execute R3
 - 3.5 For round 60 downto 79
 - 3.5.1 execute R4
4. Do addition to the block of MD and the state[5]
 - 4.1 state[0] – a
 - 4.2 state[1] – b
 - 4.3 state[2] – c
 - 4.4 state[3] – d
 - 4.5 state[4] – e

5. Empty the value stored in a, b, c, d, e and waiting for the next block of data to be processed.
6. End

- SHA1Update()

/* the function of SHA1 is to make sure the length of received data within 2^{64} and chop it into appropriate size before sent to SHA1Final */

```
void CSHA1::SHA1Update(SHA1_CTX* context, unsigned char* data, unsigned long len)
```

```
{
    unsigned int fill, left;

    left = (context->count[0] >> 3) & 0x3F;
    fill = 64 - left;

    context->count[0] += len << 3;
    context->count[0] &= 0xFFFFFFFF;
    context->count[1] += len >> 29;

    if ((context->count[0] < (len << 3))
    {
        context->count[1]++;
    }

    if ( left && len >= fill )
    {
        memcpy((void *)(context->buffer + left), (void *) data, fill);
        SHA1Transform(context, context->buffer);
        len -= fill;
        data += fill;
        left = 0;
    }

    while(len >= 64){
        SHA1Transform(context, data);
        len -=64;
        data +=64;
    }

    if(len){
        memcpy((void *)(context->buffer + left), (void *) data, len);
    }
}
```

Explanation Variables

Data – aBuffer

Len – number of characters in aBuffer

Left – how many bit still left to be processed

Fill – fill in the buffer with its number of characters

Pseudo codes

1. Start
2. If len – NULL
 - 2.1 Return control to calling function
3. Compute left by dividing count [0] with 8 and masking the result with value 64 to make sure it is smaller than 64
4. Compute fill – 64 left
5. Compute the total bits used by the data by multiplied the len with 8
6. compute the total bits to make sure whether it exceeds value 2^{32}
7. if left != NULL and len >= fill
 - 7.1 First copy the data into context-> buffer with the number of character stored in fill
 - 7.2 Then, call the SHA1Transform
 - 7.3 Len -= fill
 - 7.4 Data += fill
 - 7.5 Left = 0
8. while len >= 64
 - 8.1 Call the SHA1Transform
 - 8.2 Len -= 64
 - 8.3 Data += 64
9. if len < 64
 - 9.1 copy the data into context -> buffer with the number of character stored in len
10. end

- SHAFinal

/* The function of SHAFinal is to add padding bits and length bits in to the data in order to form a 512 bit data block. Lastly, the message digest will be computed here. */

```
void CSHA1::SHA1Final(unsigned char digest[20], SHA1_CTX* context)
{
    unsigned long i, j;
    unsigned char finalcount[8];

    for (i = 0; i < 8; i++)
    {
        finalcount[i] = (unsigned char)((context->count[(i >= 4 ? 0 : 1)] >> ((3-(i & 3)) * 8)
    & 255); /* Endian independent */
    }
    SHA1Update(context, (unsigned char *)"\200", 1);
    while ((context->count[0] & 504) != 448)
    {
        SHA1Update(context, (unsigned char *)"\0", 1);
    }

    SHA1Update(context, finalcount, 8); /* Should cause a SHA1Transform() */
    for (i = 0; i < 20; i++)
    {
        digest[i] = (unsigned char) ((context->state[i >> 2] >> ((3-(i & 3)) * 8) ) & 255);
    }

    /* Wipe variables */
    i = j = 0;
    memset(context->buffer, 0, 64);
    memset(context->state, 0, 20);
    memset(context->count, 0, 8);
    memset(&finalcount, 0, 8);
#ifdef SHA1HANDSOFF /* make SHA1Transform overwrite it's own static vars */
    SHA1Transform(context->state, context->buffer);
#endif
}
```

Explanation variables

i – counter
finalcount[8] – plaintext length

Pseudo Codes

1. start
2. for i – 0 downto 8
 - 2.1 load the length of plaintext in hexadecimal from context ->count[]
3. Call function SHA1Update(context, (unsigned char *) “\200”, 1) to add bit ‘1’ before add padding bit ‘0’
4. while the total bit in context ->buffer != 448
 - 4.1 append but ‘0’ into buffer
5. Call function SHA1Update(context, (unsigned char *) “\200”, 1) to add plaintext length (final count) and trigger SHA1Transform to form the final MD
6. Final MD in context-> state[5] is copied into digest [20]
7. All counter, variables and state is emptied
8. End

ii) Blinding Function

```
CVLong CRSA::blind( const CVLong& vPlainText, const CVLong& vMult, const  
CVLong &vPrivKey , const CVLong& vRandom)  
{  
    return modexpblind( vPlainText, vPrivKey, vMult, vRandom );  
}
```

Explanation Variables

vPlain – Plaintext
vMult - Multiple sum value
vPrivKey – Public key
vRandom – Random key

Pseudo Codes

1. Start
2. Call modexpblind function. The 4 variables are sent to the function


```
CVLong modexpblind( const CVLong & x, const CVLong & e, const CVLong & m,
const CVLong & r)
{
    CMonty me(m);
    return me.expblind( r,e,x);
}
```

Explanation Variables

x- plaintext
e- Public key
m- multiple sum value
r – random key

Pseudo Codes

1. Start
2. Call function CMonty and send multiple sum value
3. Call function expblind and send random key, public key and plaintext.

CMonty is a function that initialize the multiple sum value. This function is essential to calculate the modular value

```
CMonty::CMonty( const CVLong &M )
{
    m = M;
    N = 0; R = 1; while ( R < M ) { R += R; N += 1; }
    R1 = modinv( R-m, m );
    n1 = R - modinv( m, R );
}
```

CVLong modinv(const CVLong &a, const CVLong &m) // modular inverse
// returns i in range 1..m-1 such that $i*a = 1 \pmod m$
// a must be in range 1..m-1

```
{
    CVLong j=1,i=0,b=m,c=a,x,y;
    while ( c != 0 )
    {
        x = b / c;
        y = b - x*c;
        b = c;
        c = y;
        y = j;
        j = i - j*x;
        i = y;
    }
}
```

```

    }
    if ( i < 0 )
    {
        i += m;
    }
    return i;
}

```

expblind function is the main function to calculate the blind plaintext.

Blindtext = (plaintext)(randomkey ^ publickey) % multiple value.

```

CVLong CMonty::expblind( const CVLong &x, const CVLong &e, const CVLong &r)
{
    return (( monty_exp( (r*R)%m, e ) * R1 ) * x) % m;
}

```

monty_exp is the function that calculate the exponent value

```

CVLong CMonty::monty_exp( const CVLong &x, const CVLong &e )
{
    CVLong result = R-m, t = x;
    t.docopy();
    unsigned bits = e.value->bits();
    unsigned i = 0;
    while (1)
    {
        if ( e.value->bit(i) )
        {
            mul( result, t);
        }

        i += 1;
        if ( i == bits )
        {
            break;
        }
        mul( t, t );
    }
    return result;
}

```

Chapter 7 System Testing

7.1 Introduction

System testing is carried out in parallel throughout the software development. It is essential to check a software product meticulously after it has been developed. Testing a software product once it is ready to be implemented is far too late. This is because once the product is ready to implement, it is difficult to developer to trace the fault. Therefore, continuously testing ensures the software product is as fault free as possible at all times. Besides, system testing also is performed to determine whether the requirements that are stated at the early phase of software development are fulfilled.

7.2 Type of testing

There are a few kind of testing that the author applies while developing the program. The author uses the bottom up system testing approach which means the unit testing came first, followed by module testing and integration testing.

- Unit Testing

Unit testing deals with the smallest and most elementary units of software such as sub-function or sub-routine function. For example unit testing is performed on sub-function of the program which are ProcessSHA, Encrypt, Decrypt, Blind and etc. In unit testing which have been done throughout the development phase, is to identify the error or mistake that might be made. The testing is not only been done once but several times and each changes that has been made from the testing will also need to be tested again.

- Module Testing

Module testing is performed on the module of the blind digital signature program. In blind digital signature program, there are six modules which are blinding module, signing module, verifying module, key generator module, user authentication module and about module. Each of these modules has been tested separately by using the compiler of the Microsoft Visual C++. If there are any syntax errors the compiler will inform the author so that the author can fix the error. Although, if no errors emerge there can be run time errors that need the author to trace the fault in the source code by itself.

- Integration Testing

Integration testing is to check that the modules or components combine correctly to achieve a product that satisfies its specifications. During the integration testing, the combination of all the modules had been tested by the build tool of Microsoft Visual C++. Although there are no error on each of the modules but there can be a linking error when they are integrated to become a one module.

Chapter 8 System Evaluation

8.1 Introduction

System evaluation is a process where developer will evaluate the complete software product in terms of problems that occur during the development, the strength of the software product, the limitation of the software product and the future enhancements for the system in order to get a more satisfactory system.

8.2 System Strengths

- User Friendly

The interfaces design in blind digital signature program is very understandable. It means that user can easily understand on how to use the program. The grouping of the certain task make user understands what they need to do first. Besides, instead of the button the interfaces also provide menus and icons for the user to execute the task.

- Moderate security

Besides, this program used 128-bit key for the encryption where the security level provided is considered low to moderate.

- User Authentication

Instead of the encryption key usage, blind digital signature program also provide a security where forbidden unauthorized user to make use of the application for masquerading or fraudulent. It means this application acquired the user to authenticate itself before enter the application.

- Response time

In addition, this application takes less than 3 seconds to generate a digital signature for a document with approximately 30 000 characters.

8.3 System Limitations

- Process text file – Blind signature application only can process the document that in text file and it's own generated file such as signed and blinded file.
- Edit Box – the program cannot accommodate $2^{64} / 8$ bytes of characters. The performance of the system can be tested and evaluated at the maximum as the SHA1 only accepts the most 2^{64} bits of input.
- Key storage- this program cannot keep the keys secretly. User have to manually copy, type or store the copy in a different but removable storage medium.

8.4 Recommendation for Future enhancement

- Due to the speed of program, with hardware implementation, the time required for blinding, signing and verifying can shorten and completed in shorter CPU cycles thus increase the system performance.

- Secured key storage- A key storage with proper encryption or password protected can be developed to allow users keep their key systematically and safely.
- Higher key length- Develop a program that can support higher key length where the more higher the key length the more higher level of security as penetrator requires longer time to break through encryption.
- Unblind- Hopefully, in a future the program can be develop with the unblind function that can extract the valid digital signature from the blind signature. Besides, the unblind process also can change back the unblind message to the original message.

8.5 Problem Discussions and solutions

In this section, the problem with the solutions that the author faced will be discussed and listed below. But not all the problems that the author faced come up with solutions. In this case the author decides to discuss and gives some example to proof that the author does not meet the solutions.

1. Due to limited knowledge on using the Microsoft Foundation Classes (MFC) which is the built in function that Visual C ++ provided, the author who is the first timer has a big problem while relating the interfaces with the coding that determines the action that it should take. Furthermore, because of so limited people knows about the MFC the author have to learn from the book by itself. Unfortunately, the author also unable to get useful books as guidance due to the limited resources.

Solution:

One of the solutions is, the author has to buy the books by itself and have to spend a lot of times by doing research on how to use the MFC.

2. Blind signature is a new concept that hardly to understand. In addition, there are so limited resources whether on the internet or books that the author can refer as guidance. All sources on the internet are same and no test data given on the calculation. For example, for the unblind process, the formula that stated cannot be proven. Below are the sample of test data that indicates the problems.

Key Generation

Select two primes p and q :

$$\begin{aligned}p &= 5, q = 13 \\n &= p \cdot q \\&= (5)(13) \\&= 65\end{aligned}$$

$$\begin{aligned}(p-1)(q-1) &= \Phi(n) \\(5-1)(13-1) &= 48\end{aligned}$$

Choose number e between 5 and 48, that relatively prime to 48:

$$e = 11;$$

Choose d , such that $ed \bmod (p-1)(q-1) = 1$:

$$\begin{aligned}ed \bmod 48 &= 1 \\11(d) \bmod 48 &= 1\end{aligned}$$

$$d = 35$$

$$\begin{aligned}(11)(35) \bmod 48 &= 1 \\(11)(35) &= 385 \\385 \bmod 48 &= 1\end{aligned}$$

Therefore,

$$\begin{aligned}\text{Public key } (e, n) &: (11, 48) \\ \text{Private key } (p, q, d) &: (5, 13, 35)\end{aligned}$$

Blinding process

giving $m = 2$; $r = 5$;

$$\begin{aligned}m' &= (r^e)(m) \pmod{n} \\&= (5^{11})(2) \pmod{65} \\&= 55\end{aligned}$$

Signing process

$$\begin{aligned}s' &= (m'^d) \pmod{n} \\&= [(r^e)(m)]^d \pmod{n} \\&= (r^{ed})(m^d) \pmod{n} \\ed &= 1; \\&= r(m^d) \pmod{n} \\&= (2^{35})(5) \pmod{65} \\&= 35\end{aligned}$$

Unblinding process

Unblinding process is a process that extract the appropriate signature, s , for the original message, m , from the signature that generate on blinded message, m' . After the blinding process, the author supposedly has a valid digital signature, s , on the original message value, m .

$$\begin{aligned}s &= s'/r \pmod{n} \\&= 35/5 \pmod{65} \\&= 7\end{aligned}$$

Verifying process

The problem emerges on verifying process where the original message value cannot get it back. As for digital signature, to know whether the signature is indeed valid we have to do a comparison between the verifying result and the original message value. If both of the values are same, the signature is valid but if it is not the signature is not valid. Due to the blind signature, after the unblinding process the author supposedly to have a valid digital signature but unfortunately the author cannot proof the validity of the signature. Theoretically, the message value, m , is 2 but what the author get here is 28.

$$\begin{aligned}m &= (s^e) \pmod{n} \\&= 7^{11} \pmod{65} \\&= 28\end{aligned}$$

Solution:

As for this problem, the author does not find a good solution even trying so hard. Due to that, this blind signature program does not provide the unblinding function. The program only can verify the signature on blinded message instead of the original message.

3. At the testing stage, the author unable to provide a complete blind signature test data to the supervisor and moderator.

Solutions

As the mathematic calculations involved were too complicated, the author separated the testing data into two distinct parts. The author used different inputs for each for the ease of calculation.

References

- http://searchsecurity.techtarget.com/sDefinition/0,,sid14_gci214431,00.html
- <http://www.garykessler.net/library/crypto.html#fig01>
- <http://www.x5.net/faqs/crypto/q94.html>
- <http://www.rsasecurity.com/rsalabs/node.asp?id=2252>
- http://searchwebservices.techtarget.com/sDefinition/0,,sid26_gci212415,00.html
- http://searchvb.techtarget.com/sDefinition/0,,sid8_gci213309,00.html
- <http://www.votehere.net/remotevote.html>
- <http://www.artisoft.com/fileassurity.htm>
- <http://www.verisign.com/products-services/security-services/code-signing/digital-ids-code-signing/index.html>
- <http://www.elock.com/products/prosigner/>
- <http://ntrg.cs.tcd.ie/mepeirce/Project/double.html>
- <http://www.rsasecurity.com/rsalabs/node.asp?id=2339>
- http://www.fact-index.com/b/bl/blind_signature.html
- http://www.ici.ro/ici/revista/sic2003_4/art2.pdf

University of Malaya

Bibliography

Tan Loo Geck. (2003). *Implementing Digital Signature Method for Secure Business Transaction*. Thesis. University of Malaya.

Mohan Atreya. (2002). *Digital Signatures*. New York : MacGraw-Hill/Osborne

Ed Tittle, Mike chapple, James Michael Stewart. (2003). *CISSP: Certified Information Systems Security Professional Study Guide*. Florida : Sybex.

Suranjan Choudary, Karthik Bhetnagar, Wasim Haque and NITT. (2002). *Public Key Infrastructure Implementation and Design* : John Wiley & sons.

Nadir Gulzar and Kartik Ganeshan. (2003). *Practical J2EE Application Architecture*. New York : MacGraw-Hill/Osborne.

William Stallings. (2000). *Network Security Essentials : Applications and Standards*. New Jersey: Prentice Hall.

William Stallings. (2003). *Cryptography and Network Security : Principles and Practies* 3rd . New Jersey: Prentice Hall.

Stephen R.Schach. (2002). *Object Oriented and Classical Software Engineering*. New York : MacGraw-Hill/Osborne.

- A. Appendix**
- B. RSA Test Data**
- C. Source Code**
- D. User Manual**

University of Malaya

Appendix B

University of Malaya

RSA Test Data

Key Generation

Select two primes p and q :

$$\begin{aligned}p &= 5, q = 13 \\n &= p \cdot q \\&= (5)(13) \\&= 65\end{aligned}$$

$$\begin{aligned}(p-1)(q-1) &= \Phi(n) \\(5-1)(13-1) &= 48\end{aligned}$$

Choose number e between 5 and 48, that relatively prime to 48:

$$e = 11;$$

Choose d , such that $ed \bmod (p-1)(q-1) = 1$:

$$\begin{aligned}ed \bmod 48 &= 1 \\11(d) \bmod 48 &= 1 \\d &= 35 \\(11)(35) \bmod 48 &= 1 \\(11)(35) &= 385 \\385 \bmod 48 &= 1\end{aligned}$$

Therefore,

$$\begin{aligned}\text{Public key } (e, n) &: (11, 48) \\ \text{Private key } (p, q, d) &: (5, 13, 35)\end{aligned}$$

Blinding process

giving $m = 2$; $r = 5$;

$$\begin{aligned}m' &= (r^e)(m) \bmod n \\&= (5^{11})(2) \bmod 65 \\&= 55\end{aligned}$$

Signing process

$$\begin{aligned}s' &= (m'^d) \pmod n \\&= [(r^e)(m)]^d \pmod n \\&= (r^{ed})(m^d) \pmod n \\ed &= 1; \\&= r(m^d) \pmod n \\&= (2^{35})(5) \pmod{65} \\&= 35\end{aligned}$$

Unblinding process

$$\begin{aligned}m &= (m'/r) \pmod n \\&= (mr^e)/r \pmod n \\&= mr^{1/e} \pmod n \\&= (2)(5^{1/11}) \pmod{65} \\&= 2\end{aligned}$$

Verifying process

$$\begin{aligned}m' &= (s'^e) \pmod n \\&= 35^{11} \pmod{65} \\&= 55\end{aligned}$$

Appendix C

University of Malaya

Key generator Module

```
/* Generate two primes numbers*/
void CRSA::createKey( CVLong& vMult, CVLong& vEncKey, CVLong& vPrivKey )
{
    // Choose primes
    CVLong          vPrime1;
    CVLong          vPrime2;
    CPrimeFactory   objPrimeFact;
    // need to randomly generate two numbers -
    srand( (unsigned)( time( NULL ) ) );
    unsigned   u1= rand();

    srand( u1 );
    CVLong   vRand1 = u1 * rand();
    vPrime1 = objPrimeFact.find_prime( vRand1 );
    srand( (unsigned)(u1 * ::GetTickCount()) );
    unsigned   u2= rand();

    srand( u2 * u1 );
    CVLong   vRand2 = u2 * rand();
    vPrime2 = objPrimeFact.find_prime( vRand2 );

    if ( vPrime1 > vPrime2 )
    {
        CVLong vTmp = vPrime1;
        vPrime1 = vPrime2;
        vPrime2 = vTmp;
    }

    // Calculate public key
    vMult = vPrime1 * vPrime2;
    vEncKey = 50001; // must be odd since vPrime1-1 and vPrime2-1 are even
    while ( gcd( vPrime1-1, vEncKey ) != 1 || gcd( vPrime2-1, vEncKey ) != 1 )
    {
        vEncKey += 2;
    }

    // Calculate the private key
    vPrivKey = modinv( vEncKey, (vPrime1 - 1) * (vPrime2 - 1) );

    // Calculate the random key
    vRandomKey = 2+rand()%100;
}
```

/* Fermat Theorem. A number is probably prime if it fulfills the requirements of this mathematical equation of the theorem*/

```
int CPrimeFactory::fermat_is_probable_prime( const CVLong &p )
{
    // Test based on Fermats theorem  $a^{p-1} = 1 \pmod p$  for prime p
    // For 1000 bit numbers this can take quite a while
    const rep = 4;
    const unsigned any[rep] = { 2,3,5,7 /*,11,13,17,19,23,29,31,37..*/ };
    for ( unsigned i=0; i<rep; i+=1 )
    { if ( modexp( any[i], p-1, p ) != 1 )
        { return 0;
        }
    }
    return 1;
}

CVLong CPrimeFactory::random( const CVLong &n )
{
    CVLong x = 0;
    while (x < n )
    {
        x = x * RAND_MAX + rand();
    } return x % n;
}
```

/* To find a prime number, it must fulfill the equation $n-1 = 2^k q$, n = prime number, q =odd */

```
int CPrimeFactory::miller_rabin_is_probable_prime( const CVLong &n )
{
    srand( (unsigned) time(0) );
    unsigned T = 100;
    CVLong w = n-1;
    unsigned v = 0;
    while ( w % 2 != 0 )
    {
        v += 1;
        w = w / 2;
    }
    for (unsigned j=1; j<=T; j+=1)
    {
        CVLong a = 1 + random( n );
        CVLong b = modexp( a, w, n );
        if ( b != 1 && b != n-1 )
        {
            unsigned i = 1;
            while (1)
            { if ( i == v )
                { return 0;
                }
            }
            b = (b*b) % n;
            if ( b == n-1 )
        }
    }
}
```



```

        {
            break;
        }

        if ( b == 1 )
        { return 0;
          } i += 1;
    }
}
}return 1;
}

```

/* To determine whether the randomly generated numbers are really prime or not by using Fermat and Miller Rabin Theorem */

```

int CPrimeFactory::is_probable_prime( const CVLong & n )
{
    return fermat_is_probable_prime(n) && miller_rabin_is_probable_prime( n );
}

```

```

CPrimeFactory::CPrimeFactory( unsigned MP )
{
    np = 0;

    // Initialise pl
    char * b = new char[MP+1]; // one extra to stop search
    for (unsigned i=0;i<=MP;i+=1)
    {
        b[i] = 1;
    }

    unsigned p = 2;
    while (1)
    {
        // skip composites
        while ( b[p] == 0 )
        {
            p += 1;
        }

        if ( p == MP )
        {
            break;
        }

        np += 1;
        // cross off multiples
    }
}

```

```

        unsigned c = p*2;
        while ( c < MP )
        {
            b[c] = 0;
            c += p;
        }
        p += 1;
    }

```

```

    pl = new unsigned[ np ];
    np = 0;
    for (p=2;p<MP;p+=1)
    {
        if ( b[p] )
        {
            pl[np] = p;
            np += 1;
        }
    }
    delete [] b;
}

```

```

CPrimeFactory::~CPrimeFactory()
{
    delete [] pl;
}

```

/* Find Prime Numbers */

CVLong CPrimeFactory::find_prime(CVLong & start)

```

{
    unsigned SS = 1000; // should be enough unless we are unlucky
    char * b = new char[SS]; // bitset of candidate primes
    unsigned tested = 0;

    while (1)
    {
        unsigned i;
        for (i=0;i<SS;i+=1)
        {
            b[i] = 1;
        }

        for (i=0;i<np;i+=1)
        {
            unsigned p = pl[i];
            unsigned r = to_unsigned(start % p); // not as fast as it should be -
            // could do with special routine

```

```

        if (r)
        {
            r = p - r;
        }

```

// cross off multiples of p

```

        while ( r < SS )
        {
            b[r] = 0;
            r += p;
        }

```

// now test candidates

for (i=0;i<SS;i+=1)

```

{ if ( b[i] )

```

```

    {
        tested += 1;

```

```

        if ( is_probable_prime(start) )

```

```

        {
            delete [] b;

```

```

            return start;

```

```

        }

```

```

    } start += 1;

```

```

}

```

```

}

```

```

}

```

int CPrimeFactory::make_prime(CVLong & r, CVLong &k, const CVLong & min_p)

// Divide out small factors or r

```

{
    k = 1;

```

```

    for (unsigned i=0;i<np;i+=1)

```

```

    {
        unsigned p = pl[i];

```

// maybe pre-computing product of several primes

// and then GCD(r,p) would be faster ?

```

    while ( r % p == 0 )

```

```

    {

```

```

        if ( r == p )

```

```

        {

```

```

            return 1; // can only happen if min_p is small

```

```

        }

```

```

        r = r / p;

```

```

        k = k * p;

```

```

    if ( r < min_p )

```

```

    {

```

```

        return 0;

```

```

    }

```

```

}

```

```

    }
    return is_probable_prime( r );
}
CVLong gcd( const CVLong &X, const CVLong &Y )
{
    CVLong x=X, y=Y;
    while (1)
    {
        if ( y == 0 )
        {
            return x;
        }

        x = x % y;
        if ( x == 0 )
        {
            return y;
        }
        y = y % x;
    }
}

```

```

CVLong modinv( const CVLong &a, const CVLong &m ) // modular inverse
// returns i in range 1..m-1 such that i*a = 1 mod m
// a must be in range 1..m-1
{
    CVLong j=1,i=0,b=m,c=a,x,y;
    while ( c != 0 )
    {
        x = b / c;
        y = b - x*c;
        b = c;
        c = y;
        y = j;
        j = i - j*x;
        i = y;
    }
    if ( i < 0 )
    {
        i += m;
    }
    return i;
}

```


Signing Module

```
CVLong CRSA::encrypt( const CVLong& vPlainText, const CVLong& vMult, const CVLong& vPrivKey )
```

```
{
    return modexp( vPlainText, vPrivKey, vMult );
}
```

```
CVLong modexp( const CVLong & x, const CVLong & e, const CVLong & m )
```

```
{
    CMonty me(m);
    return me.exp( x,e );
}
```

```
CMonty::CMonty( const CVLong &M )
```

```
{
    m = M;
    N = 0; R = 1; while ( R < M ) { R += R; N += 1; }
    R1 = modinv( R-m, m );
    n1 = R - modinv( m, R );
}
```

```
CVLong CMonty::exp( const CVLong &x, const CVLong &e )
```

```
{
    return ( monty_exp( (x*R)%m, e ) * R1 ) % m;
}
```

```
void CMonty::mul( CVLong &x, const CVLong &y )
```

```
{
    // T = x*y;
    T.value->fast_mul( *x.value, *y.value, N*2 );

    // k = ( T * n1 ) % R;
    k.value->fast_mul( *T.value, *n1.value, N );

    // x = ( T + k*m ) / R;
    x.value->fast_mul( *k.value, *m.value, N*2 );
    x += T;
    x.value->shr( N );

    if (x>=m)
    {
        x -= m;
    }
}
```

```

CVLong CMonty::monty_exp( const CVLong &x, const CVLong &e )
{
    CVLong result = R-m, t = x; // don't convert input
    t.docopy(); // careful not to modify input
    unsigned bits = e.value->bits();
    unsigned i = 0;
    while (1)
    {
        if ( e.value->bit(i) )
        {
            mul( result, t);

            i += 1;
            if ( i == bits )
            {
                break;
            }
            mul( t, t );
        }
    }
    return result; // don't convert output
}

```

Appendix D

University of Malaya

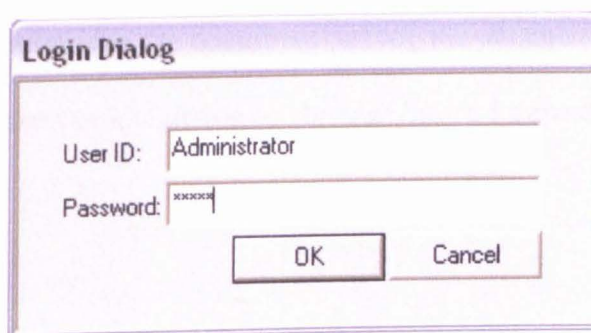
User manual

This section provides a guideline to user on how to use the blind signature application. Blind Signature application has two interfaces where the first interface only allow user to blind the document and generate the random key. The other interface let the user to generate the public key, private key and multiple sum value and also permit the user to sign the blinded document and verify the signed document. The existing of these two interfaces because there must be two kind of user which are the one who need the digital signature (user) and the one who will generate the signature (administrator) at the same time.

First of all, user needs to double click on the blind signature icon. Once the icon has been clicked, the authentication dialog box appeared. Then, user need to key in the username and password in order to enter the program. To make it easier both of the interfaces used the same username and password.

Username: Administrator

Password: Admin



The image shows a standard Windows-style login dialog box. The title bar at the top is labeled 'Login Dialog'. Inside the dialog, there are two text input fields. The first is labeled 'User ID:' and contains the text 'Administrator'. The second is labeled 'Password:' and contains masked characters represented by 'x's. Below these fields are two buttons: 'OK' and 'Cancel'.

Figure 1: Log In Dialog Box

User Interface

Once the user is authenticated, the main interface appeared. Below are the figures of the user main interface.

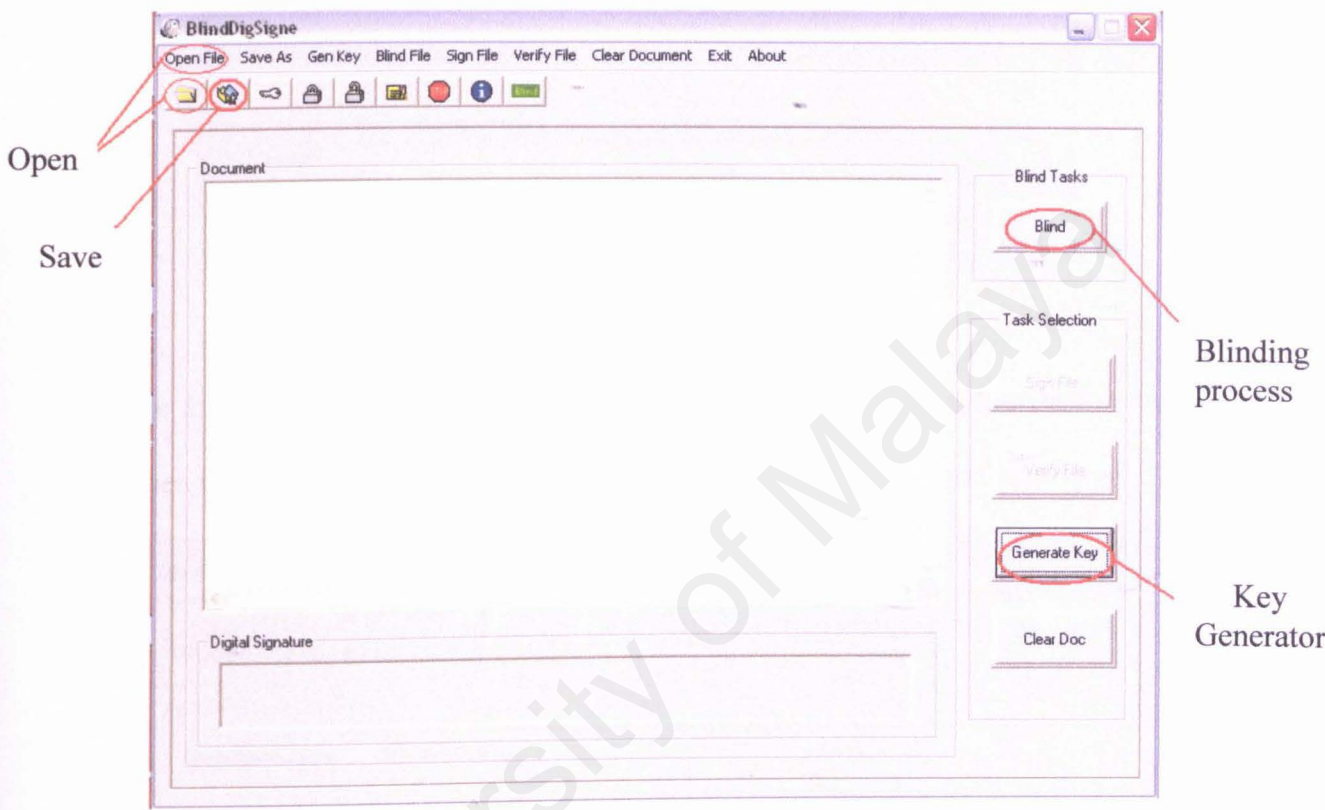


Figure 2: User main interface

Then, user need to enter a document that they intent to blind. Here, user have been given two choices whether to retrieve the document or type the document at the document part. While user clicks the open task, a dialog as show in figure 3 appeared.

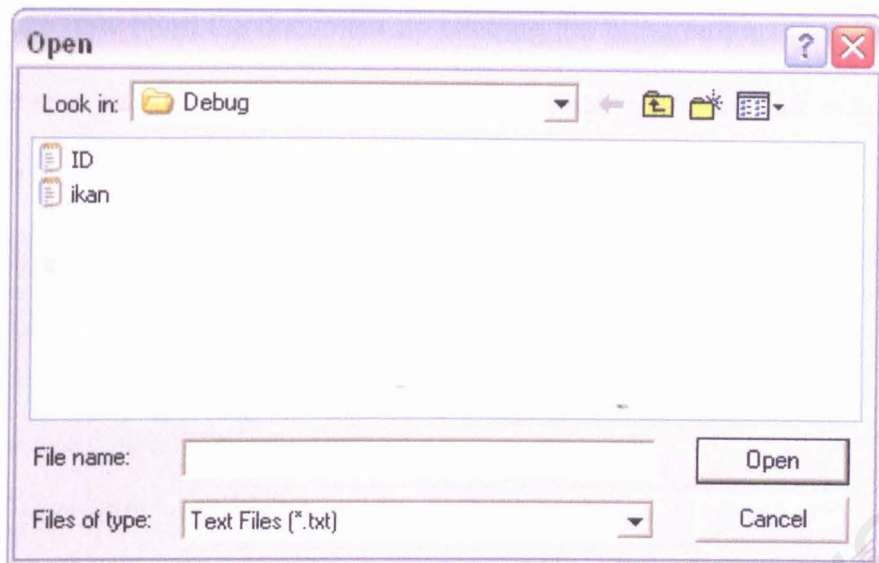


Figure 3: Open file dialog box

Once the document showed in the document part user now need to generate the random key. When user click the key generator button, figure as shown in figure 4 emerge.

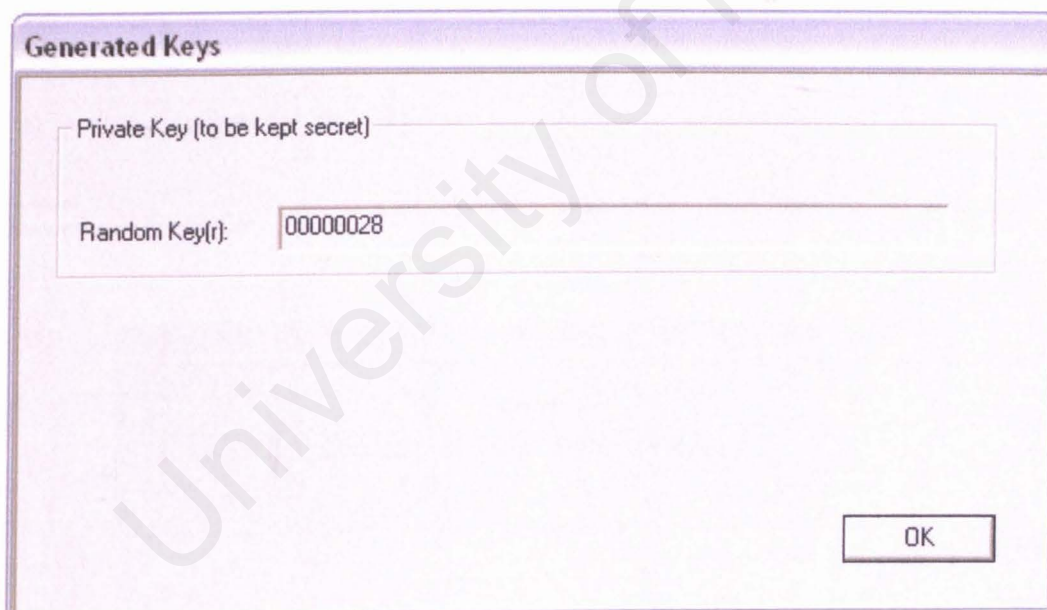


Figure 4: Random Key dialog box

Next, user can now blind the document by clicking the blind button. The public key and the multiple sum value are generated by the administrator. So, the user must get the key value before proceed to the blinding process.

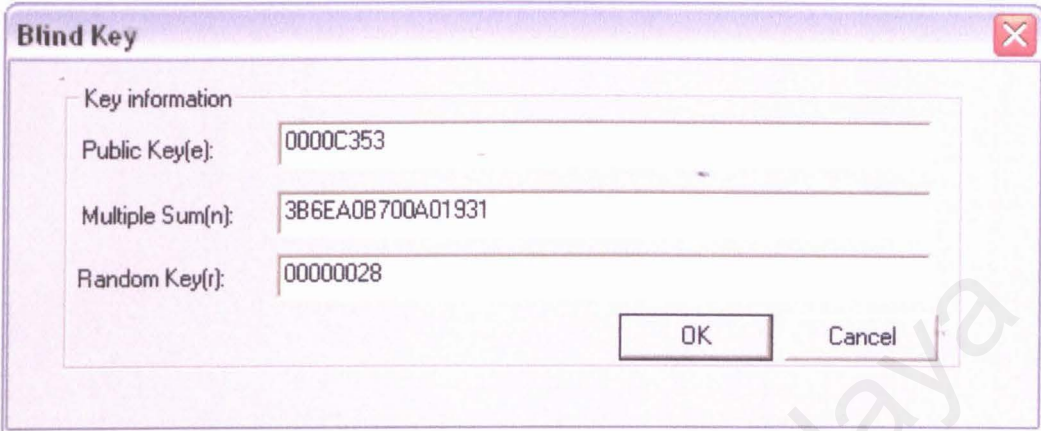


Figure 5: Blind Key dialog box

Once the OK button is clicked the blinding process proceed as shown in figure 6.

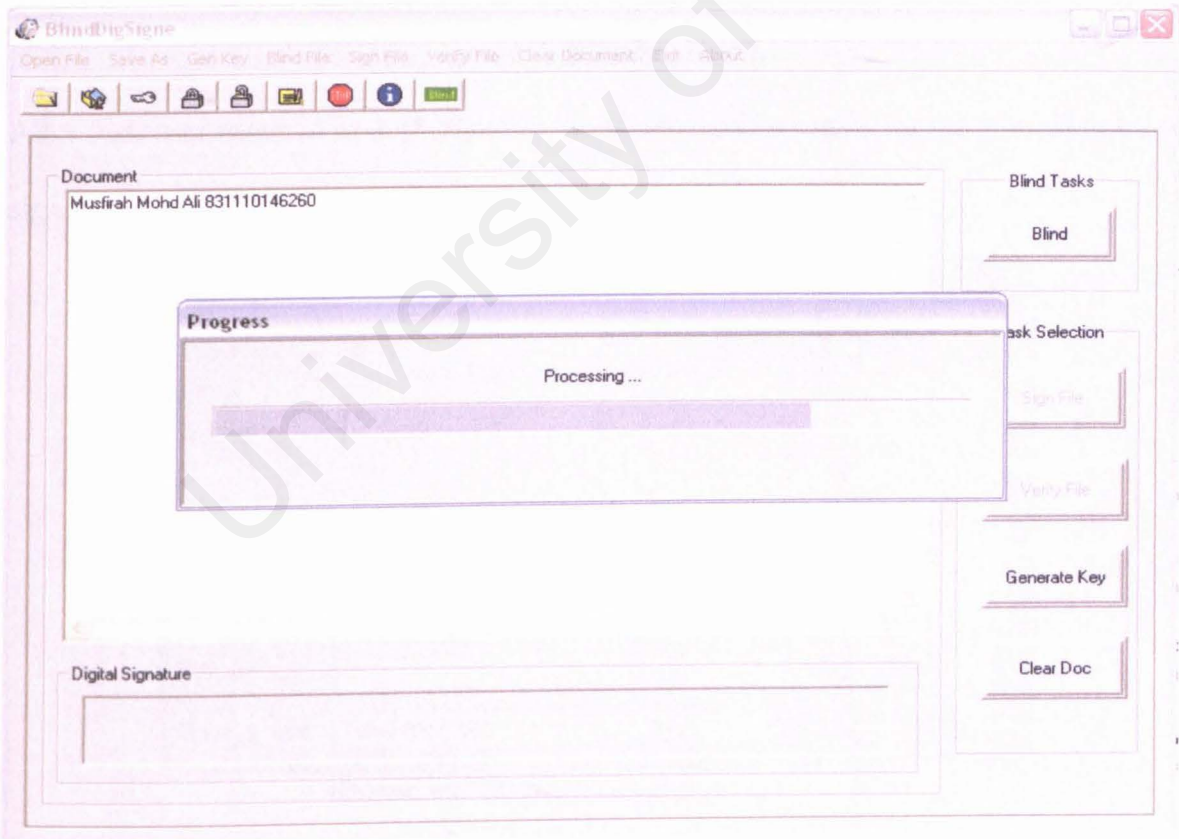


Figure 6: Blinding process progress

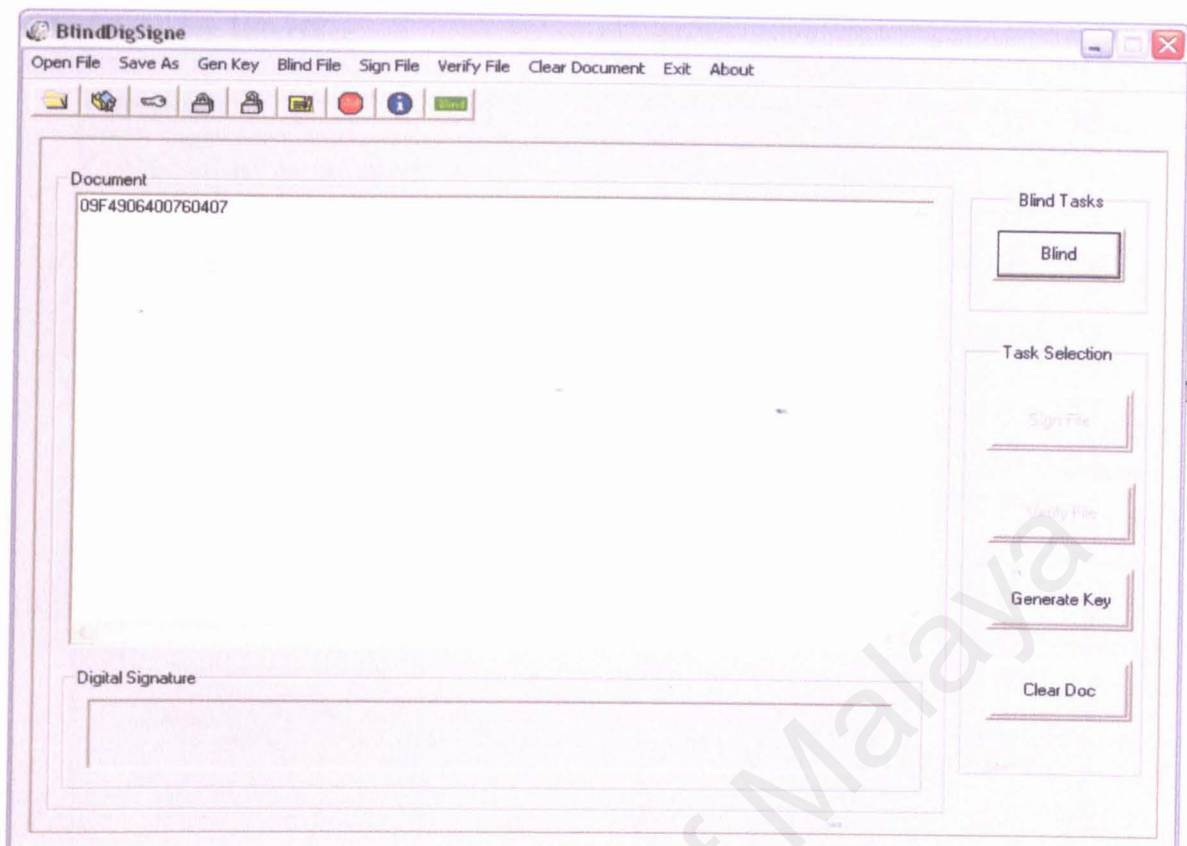


Figure 7: Blinding result

After that, user needs to save (*.blinded) the blinded document as for the administrator to sign.

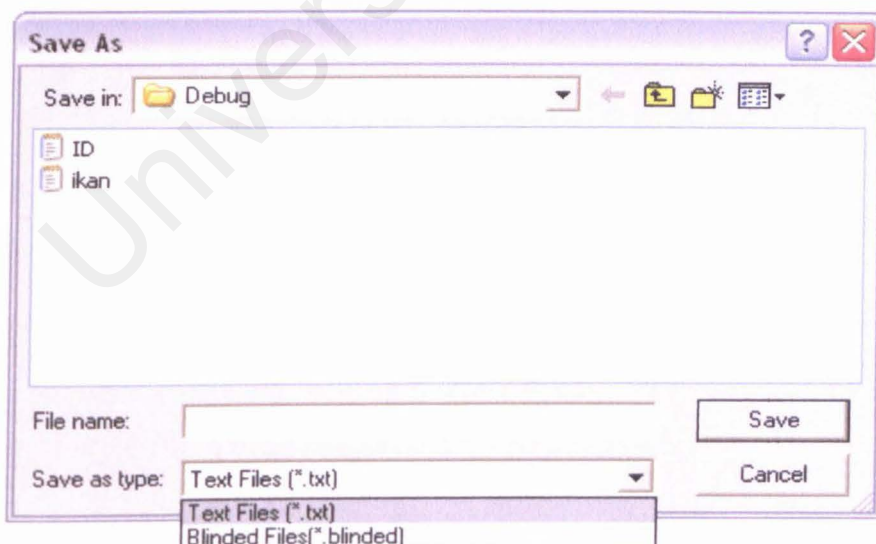
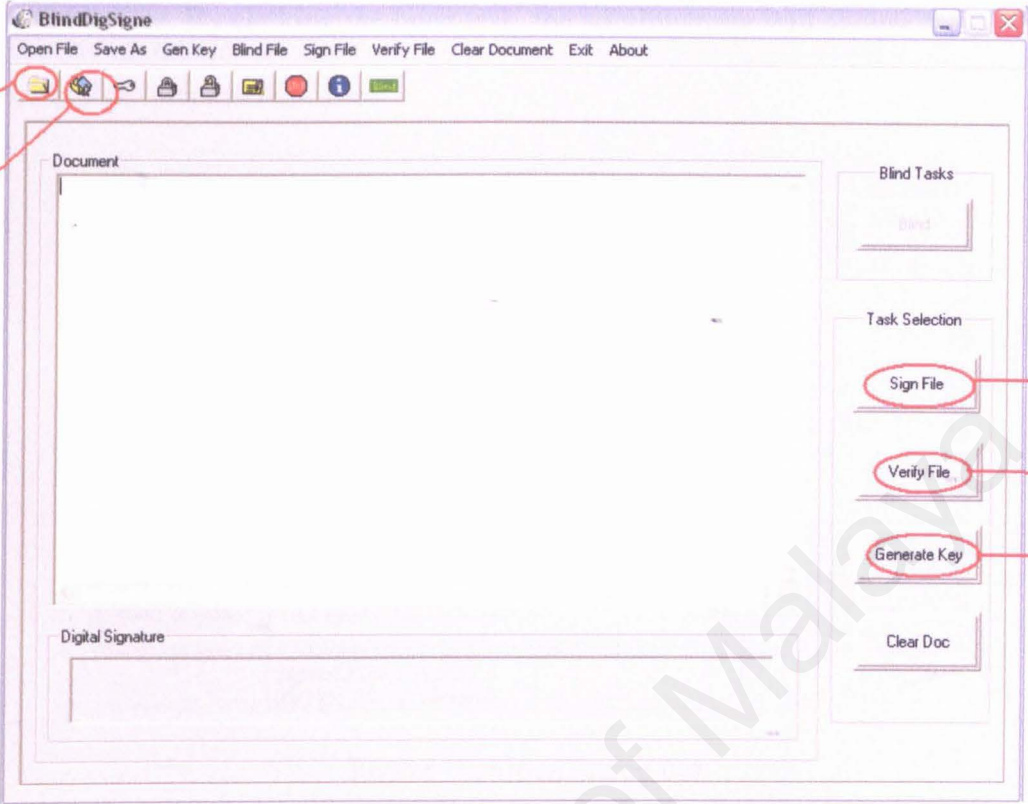


Figure 8: Save dialog box

Administrator Interface

Open

Save



Signing
Process

Verifying
Process

Key
generation

Figure 9: Administrator Main Interface

On the administrator side, administrator needs to open the blinded document that the user had saved before. By clicking the open task, the administrator will choose the blinded document that intent to be signed.

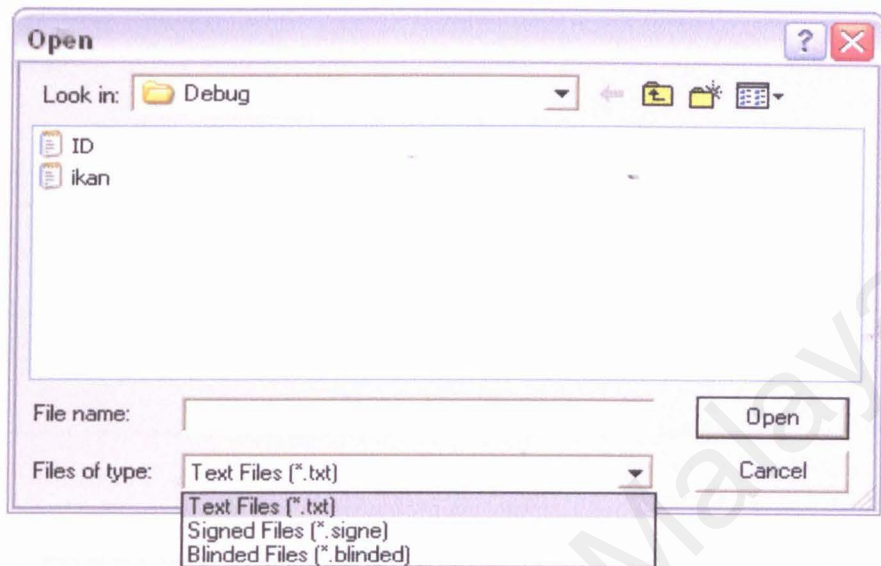


Figure 10: Open dialog box

Then, the administrator need to generate keys by clicking the key generator button as shown in figure 11.

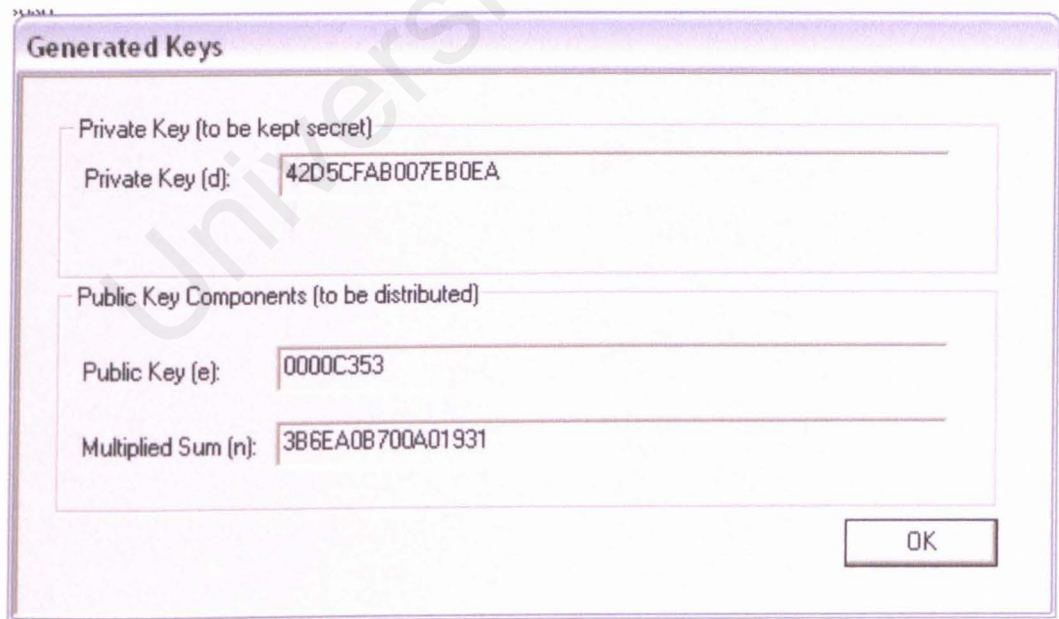


Figure 11: Key generator dialog box

Now, the administrator can proceed to sign the document by clicking the sign button.

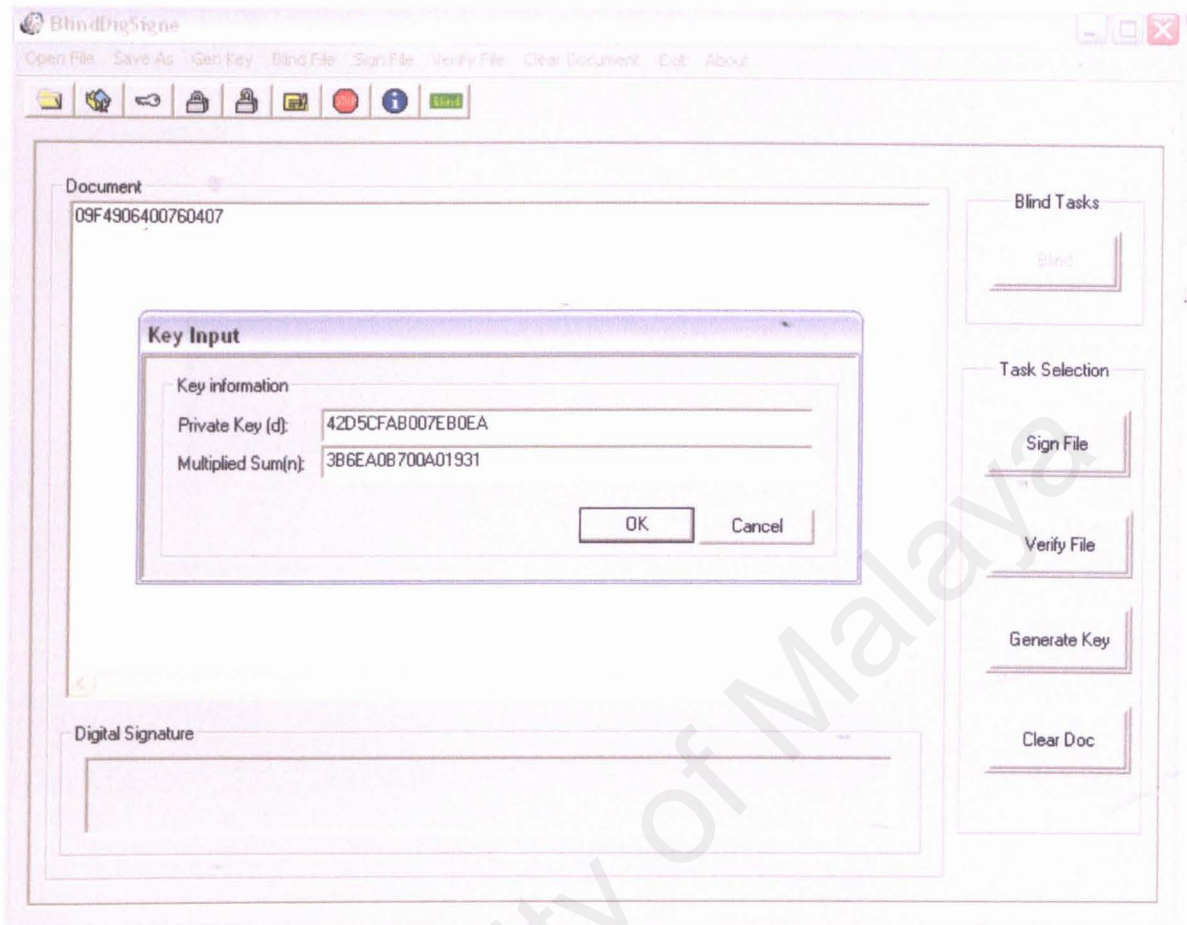


Figure 12: Signing Process

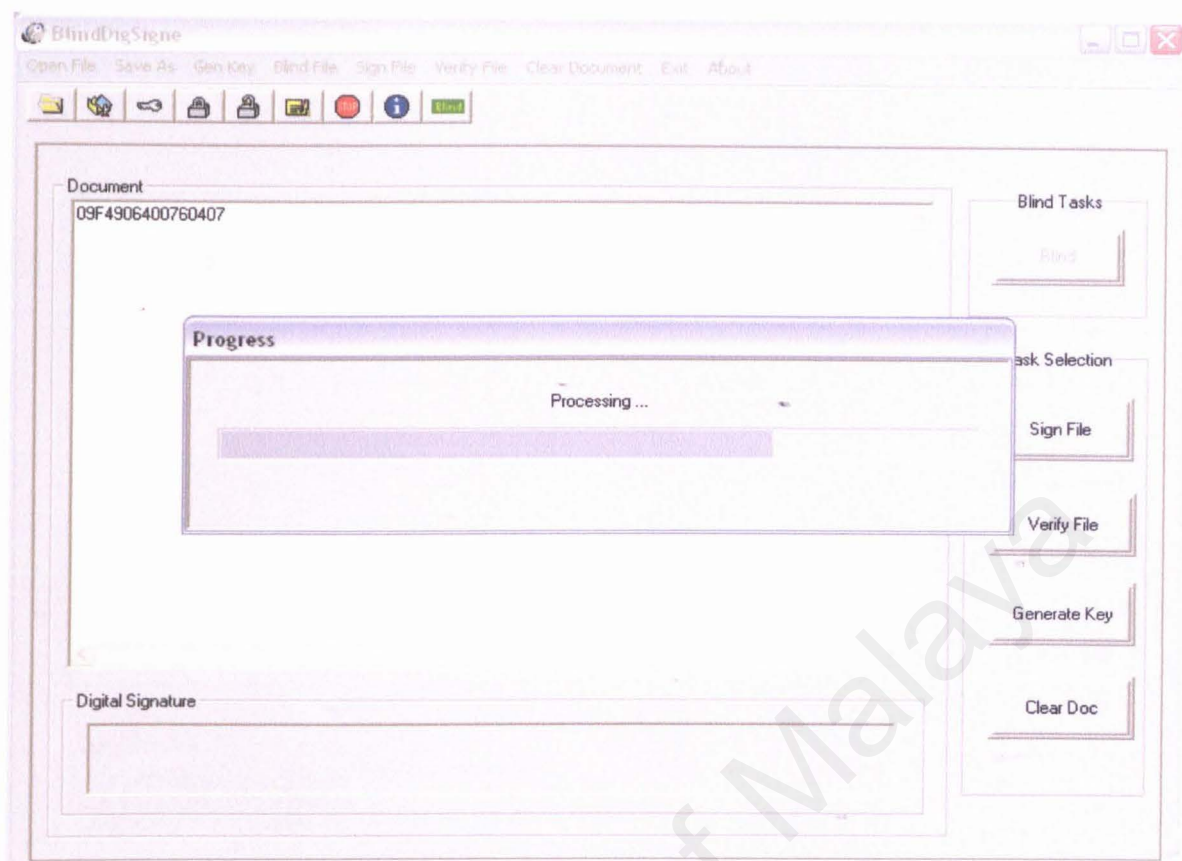


Figure 13: Signing Progress

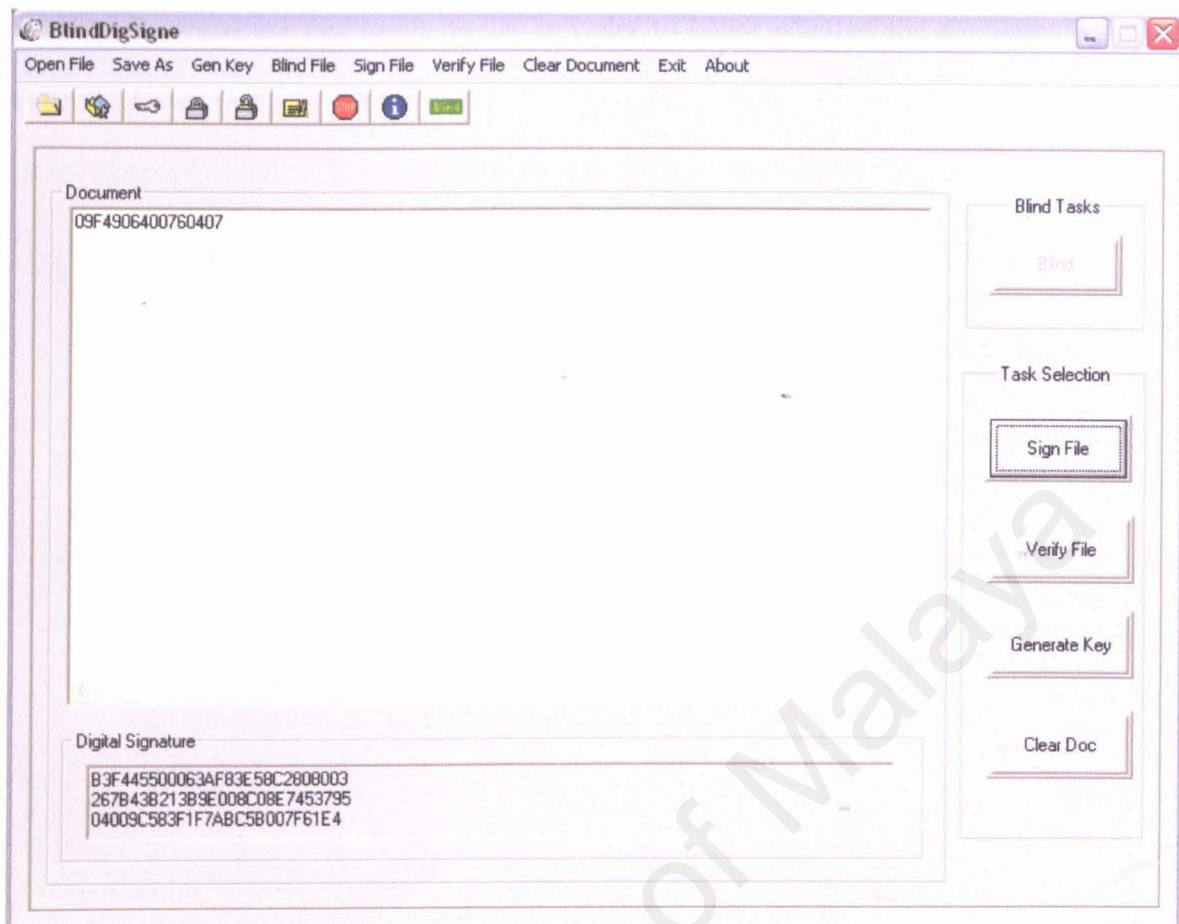


Figure 14: Signing Result

Finally, the signature can easily verify in order to know whether the signature is indeed valid as shown in figure 15.

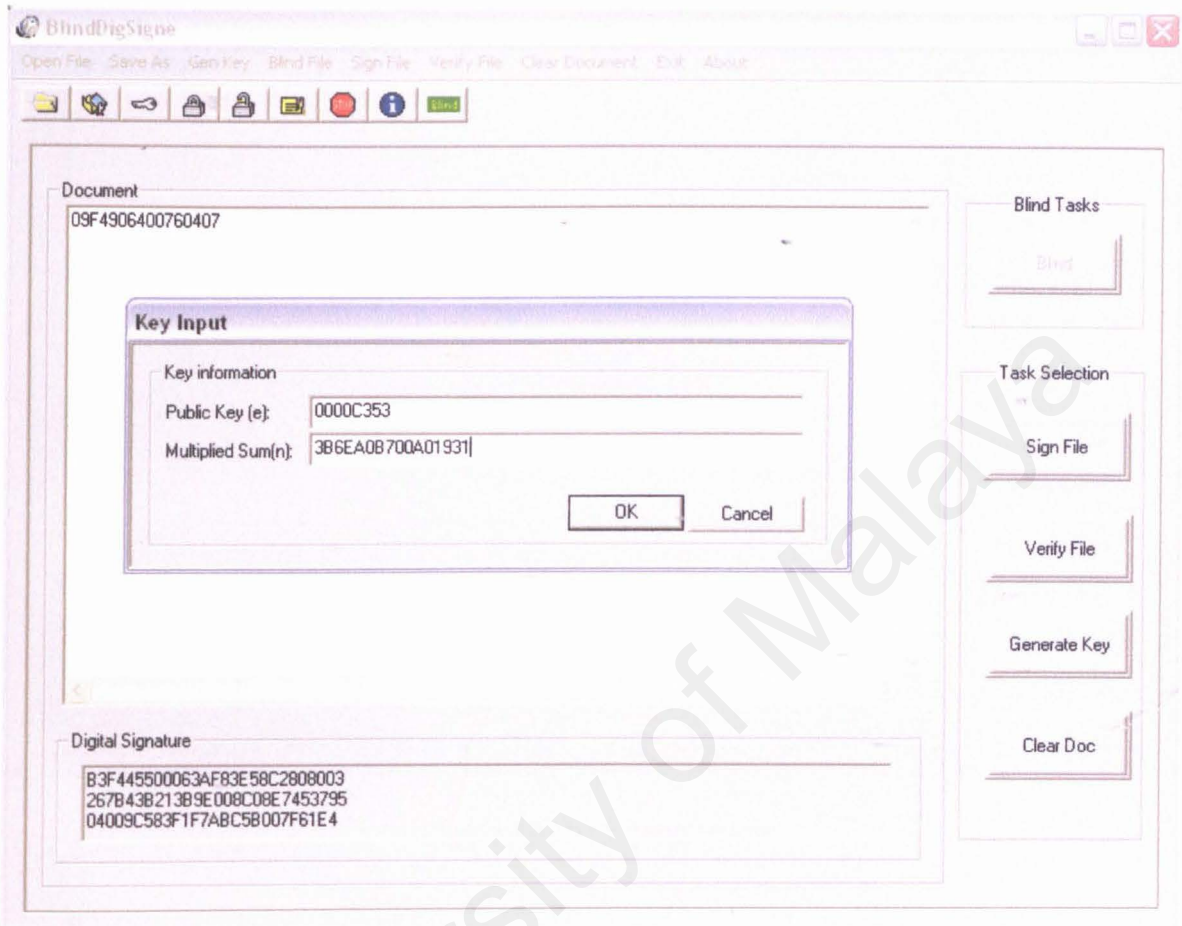


Figure 15: Verifying process

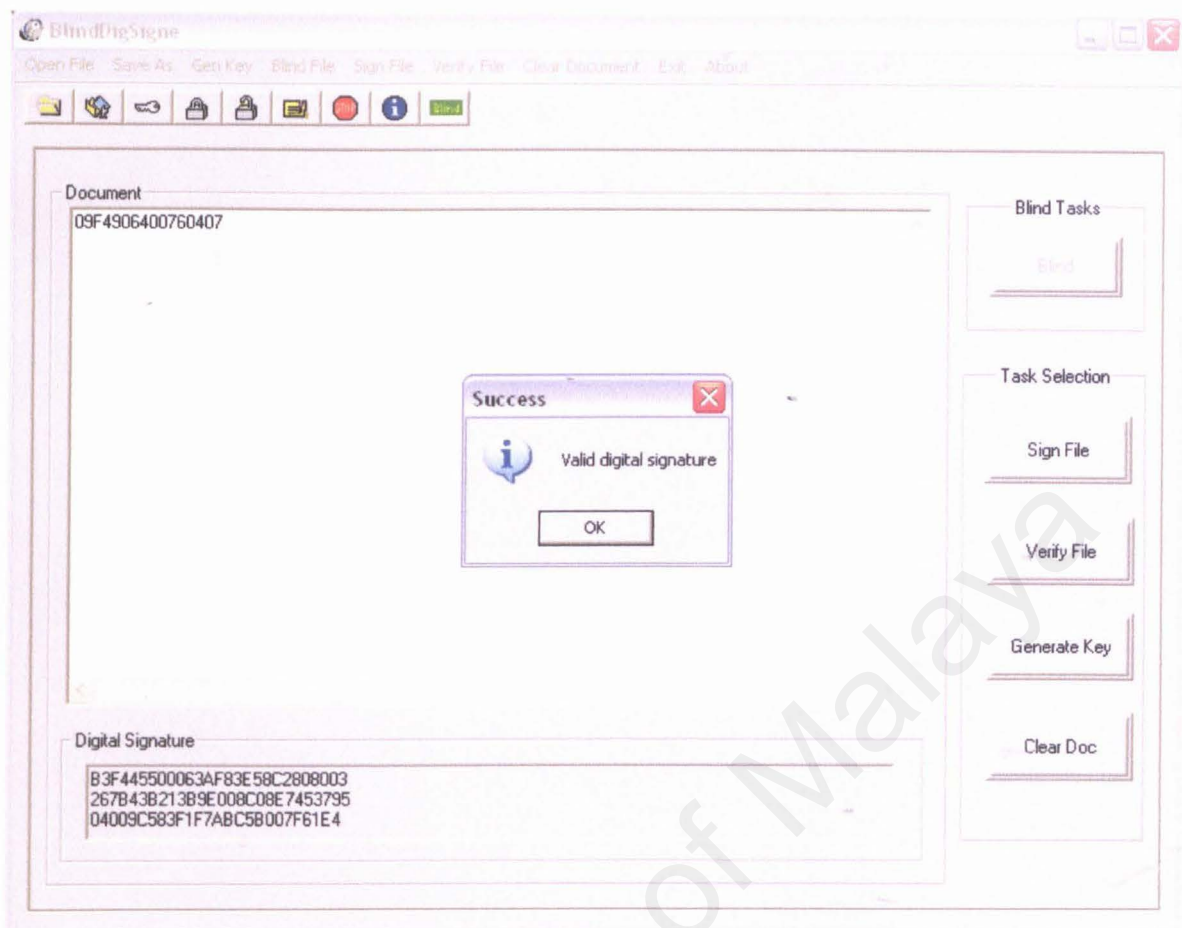


Figure 16: Verifying result